SIAM J. COMPUT.
Vol. 27, No. 1, pp. 1–47, February 1998
© 1998 Society for Industrial and Applied Mathematics

HYPERCUBIC SORTING NETWORKS*

TOM LEIGHTON† AND C. GREG PLAXTON‡

Abstract. This paper provides an analysis of a natural d-round tournament over $n = 2^d$ players and demonstrates that the tournament possesses a surprisingly strong ranking property. The ranking property of this tournament is used to design efficient sorting algorithms for several models of parallel computation:

(i) a comparator network of depth $c \cdot \lg n$, $c \approx 7.44$, that sorts the vast majority of the $n!$ possible input permutations;
(ii) an $O(\lg n)$-depth hypercubic comparator network that sorts the vast majority of permutations;
(iii) a hypercubic sorting network with nearly logarithmic depth;
(iv) an $O(\lg n)$-time randomized sorting algorithm for any hypercubic machine (other such algorithms have been previously discovered, but this algorithm has a significantly smaller failure probability than any previously known algorithm); and
(v) a randomized algorithm for sorting n $O(m)$-bit records on an $(n \lg n)$-node omega machine in $O(m + \lg n)$ bit steps.

Key words. parallel sorting, sorting networks, hypercubic machines

AMS subject classifications. 68P10, 68Q22, 68Q25, 68R05

PII. S0097539794268406

1. Introduction.

A *comparator network* is an n-input, n-output acyclic circuit made up of wires and 2-input, 2-output comparator gates. The input wires of the network are numbered from 0 to $n-1$, as are the output wires. The input to the network is an integer vector of length n where the ith component of the vector is received on input wire i, $0 \le i < n$. The two outputs of each comparator gate are labeled "min" and "max," respectively, while the two inputs are not labeled. On input x and y, a comparator gate routes $\min\{x, y\}$ to its "min" output and routes $\max\{x, y\}$ to its "max" output. It is straightforward to prove (by induction on the depth of the network) that any comparator network induces some permutation of the input vector on the n output wires. We say that a given comparator network *sorts* a particular vector if and only if the value routed to output i is less than or equal to the value routed to output $i+1$, $0 \le i < n-1$.

An n-input comparator network is a *sorting network* if and only if it sorts every possible input vector. It is straightforward to prove that any n-input comparator network that sorts the $n!$ permutations of $\{0, \ldots, n-1\}$ is a sorting network. In fact, any n-input comparator network that sorts the 2^n possible 0-1 vectors of length

* Received by the editors May 25, 1994; accepted for publication (in revised form) November 22, 1995. This article combines results that appeared in preliminary form as *A (fairly) simple circuit that usually sorts*, in Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Science Press, Los Alamitos, CA, 1990, pp. 264–274 and as *A Hypercubic sorting network with nearly logarithmic depth*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, ACM, New York, 1992, pp. 405–416.

http://www.siam.org/journals/sicomp/27-1/26840.html

† Department of Mathematics and Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 (ftl@math.mit.edu). The research of this author was supported by AFOSR contract F49620-92-J-0125 and DARPA contracts N00014-91-J-1698 and N00014-92-J-1799.

‡ Department of Computer Science, University of Texas at Austin, Austin, TX 78712 (plaxton@cs.utexas.edu). The research of this author was supported by NSF Research Initiation Award CCR-9111591 and the Texas Advanced Research Program under grant 91-003658-480. Part of this work was done while this author was visiting the MIT Laboratory for Computer Science.

n is a sorting network. The latter result is known as the 0-1 principle for sorting networks [11, section 5.3.4].

It is natural to consider the problem of constructing sorting networks of optimal depth. Note that at most $\lfloor n/2 \rfloor$ comparisons can be performed at any given level of a comparator network. Hence the well-known $\Omega(n \lg n)$ sequential lower bound for comparison-based sorting implies an $\Omega(\lg n)$ lower bound on the depth of any n-input sorting network. An elegant $O(\lg^2 n)$-depth upper bound is given by Batcher's bitonic sorting network [4]. For small values of n, the depth of bitonic sort either matches or is very close to matching that of the best constructions known (a very limited number of which are known to be optimal) [11, section 5.3.4]. Thus, one might suspect the depth of Batcher's bitonic sorting network to be optimal to within a constant factor or perhaps even to within a lower-order additive term. Consider Knuth's Exercise 5.3.4.51 (posed as an open problem; see [11]): Prove that the asymptotic value of $\hat{S}(n)$ is not $O(n \cdot \lg n)$, where $\hat{S}(n)$ denotes the minimal size (number of comparator gates) of an n-input sorting network of *any* depth. The source of the difficulty of this particular exercise was subsequently revealed by Ajtai, Komlós, and Szemerédi [2], who provided an optimal $O(\lg n)$-depth construction known as the *AKS sorting network*.

While the AKS sorting network represents a major theoretical breakthrough, it suffers from two significant shortcomings. First, the multiplicative constant hidden within the O-notation is sufficiently large that the result remains impractical. Second, the structure of the network is sufficiently "irregular" that it does not seem to map efficiently to common interconnection schemes. In fact, Cypher proved that any emulation of the AKS network on the cube-connected cycles requires $\Omega(\lg^2 n)$ time [7]. The latter issue is of significant interest, since a primary motivation for considering the problem of constructing small-depth sorting networks is to obtain a fast parallel sorting algorithm for a general-purpose parallel computer. In other words, it would be highly desirable to identify a small-depth sorting network that could be implemented efficiently on a topology that is also useful for performing operations other than sorting.

In this paper we pursue a new approach to the problem of designing small-depth sorting networks with "regular" structure. Our notion of regularity is enforced by restricting the set of permutations that can be used to connect successive levels of gates in a comparator network. In particular, we say that a comparator network is *hypercubic* if and only if successive levels are connected either by a shuffle or an unshuffle (inverse shuffle) permutation. (These terms are defined more precisely in section 3.) Knuth's Exercise 5.3.4.47 [11], posed as an open problem, may be viewed as asking for the depth complexity of *shuffle-only sorting networks*, in which every pair of adjacent levels is connected by a shuffle permutation. Batcher's bitonic sort provides an $O(\lg^2 n)$ upper bound for this problem, and recently, Plaxton and Suel [17] established an $\Omega(\lg^2 n / \lg \lg n)$ lower bound. (The same lower bound holds for the class of unshuffle-only sorting networks.)

From a practical point of view, Knuth's shuffle-only requirement would seem to be overly restrictive. It is motivated by a certain correspondence between hypercubic comparator networks and the class of *hypercubic machines* (e.g., the hypercube, butterfly, cube-connected cycles, omega, and shuffle-exchange). This correspondence allows any shuffle-only comparator network to be efficiently emulated (i.e., with constant slowdown) on any hypercubic machine. (We remark that "hypercubic machines" are more commonly referred to as "hypercubic networks" [12, Chapter 3]. We prefer the term "hypercubic machines" in the present context only because we use the

term "networks" to refer to comparator networks.) However, the class of hypercubic machines is most often characterized in terms of efficient emulation of so-called "normal" hypercube algorithms [12, Chapter 3], which effectively allow the data to be either shuffled or unshuffled at each step. (More formally, a hypercube algorithm is "normal" if it satisfies the following two conditions: (i) in any given step of the computation, communication occurs across a single dimension; and (ii) in any pair of successive steps, communication occurs across an adjacent pair of dimensions.) Thus, hypercubic comparator networks, as defined above, would seem to represent the most natural class of comparator networks corresponding to hypercubic machines.

Our approach to the design of efficient hypercubic sorting networks is based on the following *d-round no-elimination tournament* defined over $n = 2^d$ players, $d \geq 0$. For $d = 0$, the tournament has 0 rounds; no matches are played. For $d > 0$, $n/2$ matches are played in the first round according to an arbitrary pairing of the n players. The next $d - 1$ rounds are defined by recursively running a no-elimination tournament among the $n/2$ winners and (in parallel) a disjoint no-elimination tournament among the $n/2$ losers. (We chose to call this a "no-elimination" tournament to contrast it with the more usual "single-elimination" or "double-elimination" formats in which a player drops out of the tournament after suffering one or two losses.)

In a no-elimination tournament, each player achieves a unique sequence of match outcomes (wins and losses, 1's and 0's) of length d. Let player i be the player that achieves a win-loss sequence corresponding to the d-bit number i; for example, in a 5-round tournament the sequence `WLLWL` would correspond to $i = 10010_2 = 18$. Assume that the outcomes of all matches are determined by an underlying total order. Further assume that there are n distinct amounts of prize money available to be assigned to the n possible outcome sequences. How should these amounts be assigned? Clearly the largest amount of money should be assigned to player $n-1 = \mathtt{W}^d$, who is guaranteed to be the best player. Similarly, the smallest prize should be awarded to player $0 = \mathtt{L}^d$. On the other hand, it is not clear how to rank the remaining $n-2$ win-loss sequences. For instance, in an 8-round tournament, should the sequence `WLWLLWLL` be rated above or below the sequence `LLLWWWWW`? Intuition and standard practice say that the player with the 5–3 record should be ranked above the player with the 3–5 record. As we will show in section 5, however, this is not true for the sequences `WLWLLWLL` and `LLLWWWWW`. In fact, we will see that the standard practice of matching and ranking players based on numbers of wins and losses is not very good. Rather we will see that it is better to match and rank players based on their precise sequences of previous wins and losses.

The analysis of section 5 not only implies that `WLWLLWLL` is a better record than `LLLWWWWW` but also provides an efficient algorithm for computing a fixed permutation π of $\{0, \ldots, n-1\}$ such that with probability at least $1 - 2^{-n^\varepsilon}$, for some constant $\varepsilon > 0$, the actual rank of all but a small, fixed subset of the players is well approximated by $\pi(i)$, $0 \leq i < n$. (See Theorem 5.1 for a more precise formulation of this result.)

Why does the no-elimination tournament admit such a strong ranking property? Intuitively, a comparison will yield the most information if it is made between players expected to be of approximately equal strength; the outcome of a match between a player whose previous record is very good and one whose previous record is very bad is essentially known in advance, and hence will normally provide very little information. The no-elimination tournament has the property that when two players meet in the ith round, they have achieved the same sequence of outcomes in two independent no-elimination tournaments T_0 and T_1 of order $i - 1$. By symmetry, exactly half of the $n!$ possible input permutations will lead to a win by the player representing T_0

and half will lead to a win by the player representing T_1.

The remainder of the paper is organized as follows. In section 2 we discuss our applications of the no-elimination tournament. In section 3 we provide a number of definitions. In section 4 we present several basic lemmas. In section 5 we analyze the sorting properties of the no-elimination tournament. Note that section 5.3 contains a number of important technical definitions related to the no-elimination tournament. In sections 6–11 we present the applications of the no-elimination tournament discussed in section 2. In section 12 we offer some concluding remarks.

2. Overview of applications. In sections 6–11 we use the strong ranking property of the no-elimination tournament to design efficient sorting algorithms for a variety of different models of parallel computation. Most of our results are probabilistic in nature; for such results, the success probability is expressed in the form

$$1 - 2^{-2^{f(d)}}$$

for some function $f(d)$. (The parameter d is equal to $\lg n$, where n is the input size.) For the purposes of this introduction, it will be convenient to define a number of substantially different levels of "high probability" in terms of the function $f(d)$. Let us say that an event occurs *with very high probability* if $f(d) = \lg d + O(1)$, *with very*^{2} *high probability* if $f(d) = \Theta(\sqrt{d})$, with *very*^{3} *high probability* if

$$f(d) = \frac{d}{2^{\Theta(\sqrt{\lg d})}}$$

with very^4 *high probability* if $f(d) = \Theta(\frac{d}{(\lg d)\cdot \lg^* d})$, and with *very*^{5} *high probability* if $f(d)$ can be set to any function that is $o(d)$. Note that an event occurs with very high probability if and only if the corresponding failure probability is polynomially small in terms of n. As it happens, all of the main probabilistic claims made in this paper hold with very^2 high probability or better. We have defined the very high probability threshold only for the purpose of contrasting the results of section 10 with those of previous authors.

We now survey the applications of sections 6–11. In section 6 we define a comparator network of depth $c \cdot \lg n$, $c \approx 7.44$ that sorts a randomly chosen input permutation with very^5 high probability (see Theorem 6.1). (We remark that this comparator network is not hypercubic. A hypercubic version of the construction is discussed in the next paragraph.) At the expense of allowing the network to fail on a small fraction of the $n!$ possible input permutations, this construction improves upon the asymptotic depth of the best previously known sorting networks by several orders of magnitude [2, 15]. We make use of the AKS construction as part of our network. However, the use of the AKS construction can be avoided at the expense of decreasing the success probability from very^5 to very^3 high. (The depth bound remains unchanged.) The topology of our very^3 high probability network is quite simple and does not make use of expanders.

In section 7 we present a hypercubic version of the construction of section 6. In particular, we define an $O(\lg n)$-depth hypercubic comparator network that sorts a randomly chosen input permutation with very^3 high probability (see Theorem 7.1). We have not calculated the constant factor within the $O(\lg n)$-depth bound, which is moderately larger than the constant of approximately 7.44 associated with our nonhypercubic construction.

In sections 8 and 9 we provide a general method for constructing a sorting network from a comparator network that sorts most permutations. More specifically, section 8 describes how to construct a (hypercubic) high-order merging network from a (hypercubic) comparator network that sorts most input permutations. In section 9, we make use of a hypercubic high-order merging network to develop a recurrence for the depth complexity of hypercubic sorting networks. The analysis of this recurrence, presented in Appendix A, yields the main nonprobabilistic claim of our paper, namely, that there exist hypercubic sorting networks of depth

$$2^{O\left(\sqrt{\lg \lg n}\right)} \cdot \lg n.$$

Note that this bound is $o(\lg^{1+\varepsilon} n)$ for any constant $\varepsilon > 0$. (See Theorem 9.1 for a more precise form of the upper bound.) Given the aforementioned $\Omega(\lg^2 n / \lg \lg n)$ lower bound of Plaxton and Suel [17], our upper bound establishes a surprisingly strong separation between the power of shuffle-only comparator networks and that of hypercubic comparator networks.

Unfortunately, each of the network constructions given in sections 6, 7, and 9 is nonuniform in the following sense: No deterministic polynomial-time algorithm is known for generating the family of networks for which existence has been established. On the positive side, existence of a randomized polynomial-time generation algorithm for each of these network families is a straightforward consequence of our results.

In section 10, an optimal $O(\lg n)$-time randomized sorting algorithm is given for any hypercubic machine. The algorithm runs in $O(\lg n)$ time on every input permutation with very[4] high probability and uses only $O(1)$ storage at each processor. Furthermore, a very[2] high probability version of the algorithm has no more than 2 records at the same processor (where the "2" is only necessary for implementing compare-interchange operations) and requires essentially no auxiliary variables. (A global OR operation involving a single bit at each processor is used to check whether or not the sort has been completed.) A number of optimal-time randomized sorting algorithms were previously known for certain hypercubic machines. For example, the Flashsort algorithm of Reif and Valiant [19] is in this category. However, none of these algorithms has a success probability better than "very high." Probability of failure aside, Flashsort requires more storage than our algorithm, since it makes use of a $\Theta(\lg n)$-sized priority queue at each processor. On the other hand, a very high probability sorting algorithm with constant size queues has been given previously by Leighton et al. [13]. Like Batcher's $O(\lg^2 n)$ bitonic sorting algorithm, the very[2] high probability version of our sorting algorithm is *nonadaptive* in the sense that it can be described solely in terms of oblivious routing and compare-interchange operations; there is no queueing. (The very[4] high probability version is adaptive because it makes use of the Sharesort algorithm of Cypher and Plaxton as a subroutine [9].)

Note that the permutation routing problem, in which each processor has a packet of information to send to another processor, and no two packets are destined to the same processor, is trivially reducible to the sorting problem. (The idea is to sort the packets based on their destination addresses.) Hence, our sorting bounds also apply to that fundamental routing problem. In fact, standard reductions [12, section 3.4.3] allow us to apply our sorting algorithm to efficiently solve a variety of other routing problems as well (e.g., many-to-one routing with combining). Interestingly, all previously known optimal-time algorithms for permutation routing on hypercubic machines [13, 18, 20] are randomized, and do not achieve a success probability better than "very high." Thus, the results of section 10 provide a permutation routing

TABLE 1
Type conventions.

Symbol	Type	Symbol	Type
a, b, d, i, j, k, m, n	integer	$\mathcal{N}$	comparator network
c	real constant	α, β	binary string
f, g, h	function	ϵ	empty string
p, q	real number in $[0, 1]$	π	permutation
u, v, w, z	real number	Π	set of permutations
x, y	various	ϕ	0-1 vector
A, B, C	set	Φ	set of 0-1 vectors
E	probabilistic event	$o, \omega, O, \Theta, \Omega$	asymptotic symbol
X, Y	random variable	Σ	summation symbol
$\mathcal{D}$	probability distribution	$\gamma_c, \mu_c, \upsilon_c$	defined constant
$\mathcal{M}$	parallel machine	other Greek letters	real number/function

algorithm for hypercubic machines with a much smaller probability of failure than any previously known $O(\lg n)$-time algorithm.

Our final application is described in section 11, where we give a randomized algorithm for sorting n $O(m)$-bit records on an $(n \cdot \lg n)$-node omega machine in $O(m + \lg n)$ bit steps with very[2] high probability. This is a remarkable result in the sense that the time required for sorting is shown to be no more than a constant factor larger than the time required to examine a record (assuming, as is typical, that $m = \Omega(\lg n)$). The only previous result of this kind that does not rely on the AKS sorting network is the recent work of Aiello et al. [1], which provides a randomized bit-serial *routing* algorithm that runs in optimal time with very high probability on the hypercube. That paper does not address either the combining or sorting problems, however, and does not apply to any of the bounded-degree hypercubic machines (e.g., the butterfly, cube-connected cycles, omega, and shuffle-exchange). All previously known algorithms for routing and sorting on bounded-degree hypercubic machines, and for sorting on the hypercube, require $\Omega(\lg^2 n)$ bit steps.

A defect of the randomized sorting algorithms described in sections 10 and 11 is that each requires a table of permutation information to be precomputed and stored in the nodes of the machine before the algorithm is executed. Fortunately, this defect may be viewed as a relatively minor one since: (i) the table needs to be computed only once for a given machine size n (i.e., the same table can be used to sort all $n!$ possible input permutations in the time bounds stated above); (ii) the table occupies only a constant number of words per machine node; and (iii) there is a deterministic polynomial-time (in n) algorithm for computing the table. For the purposes of this paper, it is convenient to define such a "table-based" randomized sorting algorithm as *polynomial-time uniform* if and only if it satisfies properties (i), (ii), and (iii). All of the randomized sorting algorithms presented in this paper are polynomial-time uniform. (In fact, the tables used by our algorithms can be easily computed in NC.)

3. Definitions. In the sections that follow, we present basic definitions related to notational conventions, vectors, permutations, 0-1 vectors, (hypercubic) comparator networks, randomness, network composition, and network families. A number of definitions related to our analysis of the 0-1 no-elimination tournament are postponed until section 5.

3.1. Notational conventions. Our type conventions and defined constants are summarized in Tables 1 and 2, respectively. (We remark that primed and subscripted

TABLE 2
Constants.

Symbol	Constant
e	2.7182818...
γ_c	see equation (7)
μ_c	see equation (8)
υ_c	see equation (9)

variables have the same type as their unprimed and unsubscripted counterparts.)

The functions $\lg x$ and $\text{pow}(x)$ denote $\log_2 x$ and 2^x, respectively.

For all nonnegative integers a and i, let a_i denote bit i in the binary representation of a. (Bit 0 is the least significant bit.)

3.2. Vectors. A *d-vector*, $d \geq 0$, is an integer vector of length $\text{pow}(d)$. For any d-vector x, we index the components of x from 0 through $\text{pow}(d) - 1$ and denote the ith component $x(i)$.

A d-vector x is *sorted* if and only if $x(i) \leq x(i+1)$, $0 \leq i < \text{pow}(d) - 1$.

For any d-vector x, the ith *a-cube* of d-vector x, $0 \leq a \leq d$, $0 \leq i < \text{pow}(d-a)$, is the a-vector y such that $y(j) = x(i \cdot \text{pow}(a) + j)$, $0 \leq j < \text{pow}(a)$.

3.3. Permutations. A *permutation* π of length k, $k \geq 0$, is a vector of length k satisfying the following condition: For each i, $0 \leq i < k$, there is a j, $0 \leq j < k$, such that $\pi(j) = i$. If length-k permutation π is applied to length-k vector x, the resulting length-k vector x' is such that $x'(\pi(i)) = x(i)$, $0 \leq i < k$.

For all length-k permutations π and π', the length-k permutation obtained by applying π to π' is denoted $\pi \circ \pi'$.

A *d-permutation*, $d \geq 0$, is a permutation of length $\text{pow}(d)$.

Let $\Pi(d)$ denote the set of all $\text{pow}(d)!$ d-permutations.

For $0 \leq a \leq d$, let $\Pi(d, a)$ denote the $\text{pow}(a)!$ d-permutations π in $\Pi(d)$ such that: (i) π permutes within a-cubes; and (ii) π applies the same a-permutation within each a-cube.

The *shuffle d-permutation*, denoted $\hookleftarrow_d$, has ith component $i_{d-2} \cdots i_0 i_{d-1}$, $0 \leq i < \text{pow}(d)$. The *k-shuffle d-permutation*, denoted $\hookleftarrow_d^k$, is the d-permutation obtained by composing k shuffle d-permutations.

The *unshuffle d-permutation*, denoted $\hookrightarrow_d$, has ith component $i_0 i_{d-1} \cdots i_1$, $0 \leq i < \text{pow}(d)$ and is equal to the inverse of the shuffle d-permutation. Thus, $\hookrightarrow_d = \hookleftarrow_d^{-1}$. The *$k$-unshuffle d-permutation*, denoted $\hookrightarrow_d^k$, is the d-permutation obtained by composing k unshuffle d-permutations. Note that $\hookleftarrow_d^k = \hookrightarrow_d^{-k}$ for all k.

3.4. 0-1 Vectors. A *0-1 d-vector* is a d-vector over $\{0, 1\}$. Let $\Phi(d)$ denote the set of all $\text{pow}(\text{pow}(d))$ 0-1 d-vectors.

For $0 \leq k \leq \text{pow}(d)$, let $\Phi(d, k)$ denote the set of all

$$\binom{\text{pow}(d)}{k}$$

0-1 d-vectors with k 0's and $(\text{pow}(d) - k)$ 1's.

A 0-1 d-vector is *trivial* if and only if it belongs to $\Phi(d, 0) \cup \Phi(d, \text{pow}(d))$. (Otherwise, it is *nontrivial*.)

For any d-permutation π, and all k such that $0 \leq k \leq \text{pow}(d)$, we define the *kth 0-1 d-vector corresponding to d-permutation* π, denoted ϕ_π^k, as follows:

$$\phi_\pi^k(i) = \begin{cases} 0 & \text{if } 0 \leq \pi(i) < k, \\ 1 & \text{if } k \leq \pi(i) < \text{pow}(d). \end{cases}$$

Note that ϕ_π^k belongs to $\Phi(d, k)$.

For any d-permutation π, let $\Phi_\pi = \cup_{0 \leq k \leq \text{pow}(d)} \phi_\pi^k$.

Let ϕ be a 0-1 d-vector, i be the maximum index for which $\phi(i) = 0$ (or -1 if ϕ belongs to $\Phi(d, 0)$), and j be the minimum index for which $\phi(j) = 1$ (or pow(d) if ϕ belongs to $\Phi(d, \text{pow}(d))$). We say that ϕ has a *dirty region of size* $i-j+1$ corresponding to the sequence of components $\langle \phi(j), \ldots, \phi(i) \rangle$. Observe that ϕ is sorted if and only if $i = j - 1$. (Thus, the dirty region of a sorted 0-1 vector is defined to be empty and has size 0.)

A 0-1 d-vector is *a-sorted*, $0 \leq a \leq d$, if and only if it has a dirty region of size at most pow(a).

For nonnegative integers a and b, let $\Phi_M(a, b)$ denote the set of all 0-1 $(a + b)$-vectors ϕ such that every a-cube of ϕ is sorted.

We remark that if 0-1 d-vector ϕ is a-sorted, then the 0-1 d-vector ϕ' obtained by applying $\hookrightarrow_d^a$ to ϕ belongs to $\Phi_M(d - a, a)$. Furthermore, each $(d - a)$-cube of ϕ' has the same number of 0's to within 1.

3.5. (Hypercubic) comparator networks. This paper studies the depth complexity of certain classes of comparator networks. For the sake of brevity, we will use the term "network" to mean "comparator network" throughout the remainder of the paper.

For nonnegative integers a and d, a *depth-a d-network* consists of a disjoint *levels*, numbered from 0 to $a-1$, each of which has pow(d) associated *input* and *output wires*. (Note that every depth-0 d-network is the empty network.) The input and output wires of each level are numbered from 0 to pow$(d) - 1$. Output wire j on level i and input wire j on level $i + 1$ represent the same wire, $0 \leq i < a - 1$, $0 \leq j < \text{pow}(d)$. The level 0 input wires (resp., level $a - 1$ output wires) of a given network $\mathcal{N}$ are also referred to as the input wires (resp., output wires) of $\mathcal{N}$.

In order to complete our definition of a network, it remains only to define the structure and behavior of a single level. If $d = 0$, each level consists of a single wire, and the lone input is passed directly to the output. For $d > 0$, each level consists of two phases: a permutation phase followed by an operation phase.

In the permutation phase, some d-permutation π is applied to the pow(d) input wires of the level. We refer to the resulting ordered set of pow(d) wires as the *intermediate wires* of the level. In an execution of the permutation phase, the values received by the input wires are passed to the intermediate wires according to d-permutation π: Intermediate wire $\pi(j)$ receives its value from input wire j, $0 \leq j < \text{pow}(d)$.

In the operation phase, the values carried by the pow(d) intermediate wires are passed through a set of pow$(d - 1)$ 2-input, 2-output *gates*, numbered from 0 to pow$(d - 1) - 1$. Intermediate wires (resp., output wires) $2 \cdot j$ and $2 \cdot j + 1$ are input to (resp., output from) the jth gate of the level. There are five kinds of gates in our d-networks: "0," "1," "+," "−," and "?." The action of each of these gates is described below.

"0": On input (x, y), a "0" gate produces output (x, y).

"1": On input (x, y), a "1" gate produces output (y, x).

"+": On input (x, y), a "+" gate produces output $(\min\{x, y\}, \max\{x, y\})$.

"−": On input (x, y), a "−" gate produces output $(\max\{x, y\}, \min\{x, y\})$.
"?": On input (x, y), a "?" gate produces output (x, y) with probability 1/2, and output (y, x) with probability 1/2. This gate is only used in sections 10 and 11.

A d-network is *hypercubic* if and only if the d-permutation applied in each level of the d-network is either $\hookleftarrow_d$ or $\hookrightarrow_d$.

3.6. Randomness. A d-network $\mathcal{N}$ is *deterministic* if and only if $\mathcal{N}$ satisfies the following conditions: (i) the d-permutation applied in the permutation phase of each level is fixed; (ii) the type of each gate is fixed; and (iii) no gate is of type "?."

In general, we allow our d-networks to be random. A depth-a d-network $\mathcal{N}$ is *random* if and only if $\mathcal{N}$ is given by some fixed probability distribution over the set of all deterministic depth-a d-networks. (Each time an input vector is passed to a random network $\mathcal{N}$, the network behaves as a randomly chosen deterministic network drawn from the distribution defining $\mathcal{N}$.)

We have introduced the notion of a random network primarily as a technical convenience, since the random aspects of any construction can be eliminated using Lemma 4.8. Unfortunately, reliance on Lemma 4.8 leads to network constructions that are not polynomial-time uniform.

In sections 10 and 11, we make use of the "?" gate. A d-network $\mathcal{N}$ is *coin-tossing* if and only if $\mathcal{N}$ satisfies the following conditions: (i) the d-permutation applied in the permutation phase of each level is fixed; and (ii) the type of each gate is fixed. Note that: (i) "?" gates are allowed in a coin-tossing d-network; and (ii) every deterministic d-network is a coin-tossing d-network. (We do not consider random coin-tossing networks in any of our applications. Rather we view the "?" gate as an alternative to the form of randomness introduced above.)

A d-vector is *a-random*, $0 \le a \le d$, if and only if it is chosen from a probability distribution that assigns the same probability to any pair of d-vectors related by some d-permutation in $\Pi(d, a)$.

Let $\Pi_R(d)$ and $\Pi_R(d, a)$ denote the uniform distributions over $\Pi(d)$ and $\Pi(d, a)$, respectively.

Let $\Phi_R(d, k)$ denote the uniform distribution over $\Phi(d, k)$.

For all p in $[0, 1]$, let $\Phi_B(d, p)$ denote the distribution that assigns probability

$$p^k \cdot (1 - p)^{\mathrm{pow}(d)-k}$$

to each 0-1 d-vector in $\Phi(d, k)$. Note that a random 0-1 d-vector drawn from this distribution corresponds to the sequence of outcomes of d independent, p-biased Bernoulli trials.

If $\mathcal{D}$ (resp., $\mathcal{D}'$) is the probability distribution over $\Phi(d)$ that assigns probability p_i (resp., p'_i) to the d-bit binary string $i_{d-1} \cdots i_0$, $0 \le i < \mathrm{pow}(d) = n$, then define $\mathcal{D} \le \mathcal{D}'$ if and only if there exist real numbers x_{ij} in $[0, 1]$, $0 \le i < n$, $0 \le j < n$, such that:

(i) $\sum_{0 \le i < n} x_{ij} = 1$, $0 \le j < n$;
(ii) $p'_i = \sum_{0 \le j < n} x_{ij} \cdot p_j$, $0 \le i < n$; and
(iii) $x_{ij} > 0$ only if $i_k \ge j_k$, $0 \le k < d$.

Note that conditions (i) and (ii) ensure that $\sum_{0 \le i < n} p_i = \sum_{0 \le i < n} p'_i = 1$. Informally, $\mathcal{D} \le \mathcal{D}'$ if and only if it is possible to sample from $\mathcal{D}$ by first sampling from $\mathcal{D}'$ and then changing (according to a probability distribution that may depend on the particular sample chosen from $\mathcal{D}'$) a randomly chosen subset of the 1's to 0's.

3.7. Network composition. The main goal of this paper is to provide efficient (i.e., small-depth) constructions of d-networks having certain sorting-related properties. In simple cases, we present our constructions by explicitly specifying the permutation phase and operation phase of each level of the d-network. However, this approach would be too cumbersome for some of our more complicated constructions.

In general, we present a given d-network as the "composition" of a sequence of explicitly specified d-networks, explicitly specified d-permutations, and recursively specified d-networks. The following mechanical procedure can be used to convert such a sequence into a d-network.

1. "Unwind" the recurrence to obtain a sequence of explicitly specified d-networks and d-permutations.
2. Repeatedly apply the composition rules specified below to adjacent pairs in the sequence until the sequence has been reduced to either (i) a single d-network $\mathcal{N}$ or (ii) a d-network $\mathcal{N}$ followed by a d-permutation π. (The composition rules are easily seen to be associative; hence, the order of application is immaterial.) In case (i), $\mathcal{N}$ is the desired d-network. In case (ii), the desired d-network $\mathcal{N}'$ is obtained by appending a single level to $\mathcal{N}$, where: (i) the permutation π is applied in the permutation phase and (ii) the operation phase consists of $\text{pow}(d-1)$ "0" gates.

We now specify the three composition rules (network-network, permutation-network, and permutation-permutation) that can be applied in step 2 above.

1. Let $\mathcal{N}$ and $\mathcal{N}'$ denote d-networks of depth a and b, respectively. Then the composition of the pair $(\mathcal{N}, \mathcal{N}')$ is the depth-$(a+b)$ d-network $\mathcal{N}''$ such that: (i) the first a levels of $\mathcal{N}''$ are given by $\mathcal{N}$ and (ii) the last b levels of $\mathcal{N}''$ are given by $\mathcal{N}'$.
2. For any depth-a d-network $\mathcal{N}$ and d-permutation π, the composition of the pair $(\pi, \mathcal{N})$ is the depth-a d-network $\mathcal{N}'$ obtained from $\mathcal{N}$ by replacing the permutation π' applied in the permutation phase of level 0 with the permutation $\pi \circ \pi'$.
3. For all d-permutations π and π', the composition of the pair (π, π') is $\pi \circ \pi'$.

Our recursive d-network constructions employ recursion over a-cubes for some a such that $0 \leq a \leq d$. To achieve efficient performance, it is desirable for the the depth of a recursive construction over a-cubes to be a function of a, and not d. The following definitions are extremely helpful for establishing results of this kind.

A d-network $\mathcal{N}$ is *a-partitionable*, $0 \leq a \leq d$, if and only if $\mathcal{N}$ can be partitioned into $\text{pow}(d-a)$ disjoint a-networks $\mathcal{N}_i$, $0 \leq i < \text{pow}(d-a)$, where the input (resp., output) wires of $\mathcal{N}_i$ correspond to the ith input (resp., output) a-cube of $\mathcal{N}$.

Let $\mathcal{N}$ denote an a-partitionable d-network and define $\mathcal{N}_i$ as in the preceding paragraph, $0 \leq i < \text{pow}(d-a)$. Then $\mathcal{N}$ is a *(d,a)-network*, $0 \leq a \leq d$, if and only if the $\mathcal{N}_i$'s are all identical.

We remark that: (i) every d-network is a (d,d)-network; (ii) the set of (d,a)-networks is closed under composition; (iii) for all (d,a)-networks $\mathcal{N}$ and d-permutations π in $\Pi(d,a)$, the composition of the pair $(\pi, \mathcal{N})$ is a (d,a)-network; and (iv) Lemma 4.10 provides a useful alternative characterization of the class of a-partitionable hypercubic d-networks.

3.8. Network families. It is straightforward to prove by induction on the leveled structure of any d-network that the output d-vector is related to the input d-vector by some d-permutation. We say that a d-network is a *sorting network* if and only if, in addition, every possible input d-vector leads to a sorted output d-vector. (In the

case of a random d-network, a given input d-vector may not always produce the same output d-vector. We say that a given input vector is sorted by a random network $\mathcal{N}$ if and only if it is guaranteed to be sorted by $\mathcal{N}$.) As indicated in section 1, a d-network $\mathcal{N}$ is a sorting network if and only if: (i) $\mathcal{N}$ sorts all d-permutations in $\Pi(d)$ or (ii) $\mathcal{N}$ sorts all 0-1 d-vectors in $\Phi(d)$. Because of these facts, we will often find it useful to restrict our attention to inputs drawn from $\Pi(d)$ or $\Phi(d)$.

For any d-network $\mathcal{N}$, let $Sort(\mathcal{N})$ denote the set of all integer d-vectors sorted by $\mathcal{N}$.

For any (d, a)-network $\mathcal{N}$, let $Sort(\mathcal{N}, a)$ denote the set of all integer input d-vectors to $\mathcal{N}$ for which every output a-cube is sorted.

The *depth-a shuffle-"+" d-network* is the depth-a hypercubic d-network in which every level consists of the d-permutation $\hookleftarrow_d$ followed by a set of $\mathrm{pow}(d-1)$ "+" gates. We define other networks similarly; for example, each level of a depth-a unshuffle-"0" d-network consists of the d-permutation $\hookrightarrow_d$ followed by a set of $\mathrm{pow}(d-1)$ "0" gates.

A d-network $\mathcal{N}$ is in $Sort_N(d, \varepsilon)$ if and only if the output of $\mathcal{N}$ is sorted with probability at least $1-\varepsilon$ on any d-random 0-1 input d-vector.

A (d, a)-network $\mathcal{N}$ is in $Sort_N(d, a, \varepsilon)$ if and only if each output a-cube of $\mathcal{N}$ is sorted with probability at least $1-\varepsilon$ on any a-random 0-1 input d-vector.

A (d, a)-network $\mathcal{N}$ is in $Sort_N(d, a, b, \varepsilon)$, $0 \le b \le a$, if and only if each output a-cube of $\mathcal{N}$ is b-sorted with probability at least $1-\varepsilon$ on any a-random 0-1 input d-vector.

A 0-1 d-vector ϕ is *a-mostly-sorted* with respect to permutation π, $0 \le a \le d$, if and only if after applying d-permutation π, the length-$(\mathrm{pow}(d) - \mathrm{pow}(a))$ prefix of the resulting 0-1 d-vector is a-sorted.

A (d, a)-network $\mathcal{N}$ is in $Most_N(d, a, b, \varepsilon)$, $0 \le b \le a$, if and only if there exists a d-permutation π in $\Pi(d, a)$ such that each output a-cube of $\mathcal{N}$ is b-mostly-sorted with respect to π with probability at least $1-\varepsilon$ on any a-random 0-1 input d-vector.

For nonnegative integers a and b, an *(a, b)-merge* operation takes as input $\mathrm{pow}(b)$ sorted lists of length $\mathrm{pow}(a)$ and produces a single sorted list of length $\mathrm{pow}(a+b)$.

A $(d, a+b)$-network $\mathcal{N}$ is in $Merge_N(d, a, b)$ if and only if $\mathcal{N}$ performs an (a, b)-merge operation on each $(a+b)$-cube. We assume the following input convention within each $(a+b)$-cube A: The ith sorted input list is provided in ascending order in a-cube i of A, $0 \le i < \mathrm{pow}(b)$.

A *d-insertion* operation, $d \ge 0$, takes as input a sorted list of length $\mathrm{pow}(d) - 1$ and one additional input, and produces a sorted list of length $\mathrm{pow}(d)$.

A (d, a)-network $\mathcal{N}$ is in $Insert_N(d, a)$, $0 \le a \le d$, if and only if $\mathcal{N}$ performs an a-insertion operation on each a-cube. We assume the following input convention within each a-cube: The sorted list is provided in ascending order on input wires 0 through $\mathrm{pow}(a) - 2$ of the a-cube, and the additional input is provided on input wire $\mathrm{pow}(a) - 1$ of the a-cube.

In the preceding paragraphs we defined a number of network families. In each case, the name of the family is subscripted by the letter "N" (for "network"). For any particular family $\mathcal{F}_N$, we define $\mathcal{F}_D$ as the minimum depth of any network in $\mathcal{F}_N$. (For example, $Sort_D(d, a, b, \varepsilon)$ denotes the minimum depth of any network in $Sort_N(d, a, b, \varepsilon)$.) Furthermore, we let $\mathcal{F}_N^h$ denote the set of all hypercubic networks in $\mathcal{F}_N$, and $\mathcal{F}_D^h$ denote the minimum depth of any network in $\mathcal{F}_N^h$.

4. Basic lemmas. In this section, we present a number of basic lemmas.

LEMMA 4.1. *A d-network $\mathcal{N}$ is a sorting network if and only if*

$$\Phi(d) \subseteq Sort(\mathcal{N}).$$

Proof. This lemma is known as the 0-1 principle for sorting networks and is proven in [11, section 5.3.4]. (The proof is given in the context of deterministic networks. The extension to random networks is immediate, however, since a random network $\mathcal{N}$ is a sorting network if and only if every deterministic network that is assigned a nonzero probability by the distribution associated with $\mathcal{N}$ is a sorting network.) □

LEMMA 4.2. *For any d-network $\mathcal{N}$ and d-permutation π, we have*

$$\pi \in \mathit{Sort}(\mathcal{N}) \Longleftrightarrow \Phi_\pi \subseteq \mathit{Sort}(\mathcal{N}).$$

Proof. This result follows from a slight modification to the proof of the 0-1 principle cited above. □

LEMMA 4.3. *For all a and d such that $0 < a \leq d$, we have*

$$\mathit{Merge}_D(d, a-1, 1) \leq a,$$
$$\mathit{Merge}^h_D(d, a-1, 1) = O(a).$$

Proof. These bounds are established by Batcher's bitonic merge network [4]. For a hypercubic construction, additional depth is required in order to conform with the input and output conventions adopted in section 3. (Our input and output conventions have been chosen to simplify the presentation and not to minimize the constant factors associated with our hypercubic constructions.) This is accomplished by preceding Batcher's bitonic merge network with an appropriate fixed permutation π. (Note that such a fixed permutation does not contribute to the depth of a nonhypercubic construction.)

The role of the fixed permutation π is twofold: (i) to reverse one of the two sorted input lists within each a-cube, as required by Batcher's bitonic merge, and (ii) to "compensate" for the series of a shuffle permutations accompanying the merge (as described below). It is straightforward to implement an appropriate permutation π with an $O(a)$-depth hypercubic d-network. (We could use Lemma 4.7 for this purpose, although the permutation π is simple enough to implement directly.) A depth-a shuffle-"+" d-network can then be used to implement Batcher's bitonic merge network within each a-cube. □

LEMMA 4.4. *For all a and d such that $0 \leq a \leq d$, we have*

$$\mathit{Insert}_D(d, a) \leq a,$$
$$\mathit{Insert}^h_D(d, a) = O(a).$$

Proof. This bound is also established by Batcher's bitonic merge network [4]. In contrast with the construction of Lemma 4.3, no list reversal is required since the input is already in bitonic form. (We remark that in the classic sorting network model, where a given level may contain fewer than $\mathrm{pow}(d-1)$ gates, it is possible to match this depth bound while achieving size $\mathrm{pow}(d)-1$ instead of $d \cdot \mathrm{pow}(d)$ [14] (also see [12, section 3.5.4]). The basic idea is to use a tree-like network.) □

LEMMA 4.5. *For all a and d such that $0 \leq a \leq d$, we have*

$$\mathit{Sort}^h_D(d, a, 0) = O(a^2).$$

Proof. This bound is due to Batcher [4] and follows from repeated application of Lemma 4.3. □

LEMMA 4.6. *For all a and d such that $0 \leq a \leq d$, we have*

$$Sort_D(d, a, 0) = O(a).$$

Proof. This bound is due to Ajtai, Komlós, and Szemerédi [2]. Unfortunately, the constant factor associated with the AKS sorting network is impractically large. □

LEMMA 4.7. *For each d-permutation π in $\Pi(d, a)$ there is a hypercubic (d, a)-network $\mathcal{N}$ with the following properties:* (i) *$\mathcal{N}$ implements the permutation π;* (ii) *$\mathcal{N}$ has depth exactly $2 \cdot a$;* (iii) *the d-permutation $\hookrightarrow_d$ is applied in the permutation phase of each of the first a levels of $\mathcal{N}$;* (iv) *the d-permutation $\leftrightarrow_d$ is applied in the permutation phase of each of the last a levels of $\mathcal{N}$; and* (v) *every gate of $\mathcal{N}$ is either* "0" *or* "1." *Furthermore, the gate assignments of $\mathcal{N}$ can be computed in time polynomial in* $\mathrm{pow}(a)$.

Proof. This is a straightforward consequence of the work of Beneš [5]. In particular, for $a = d$, the Beneš permutation network corresponds to a hypercubic d-network satisfying properties (i), (iii), and (v). Furthermore, properties (ii) and (iv) are very nearly satisfied by the same construction; the d-network has depth $2 \cdot d - 1$ and applies the d-permutation $\leftrightarrow_d$ in the last $d - 1$ levels. It follows trivially that the claim of the lemma holds for $a = d$. (We can simply append a dummy shuffle level to the depth-$(2 \cdot d - 1)$ Beneš permutation network corresponding to an unshuffled version of π.)

For implementing a permutation in $\Pi(d, a)$, we apply our modified Beneš construction to each a-cube via the corresponding (d, a)-network of depth $2 \cdot a$. (The original Beneš construction could not be used in this manner; for $0 < a < d$, it would not map input a-cubes to output a-cubes.) □

LEMMA 4.8. *Let $\mathcal{D}$ denote an arbitrary probability distribution over the set of all d-vectors and $\mathcal{N}$ denote a random coin-tossing depth-a d-network. If $\mathcal{N}$ sorts a random d-vector drawn from $\mathcal{D}$ with probability at least p, then there exists some deterministic depth-a d-network $\mathcal{N}'$ with the same property.*

Proof. A simple averaging argument. (Note that $\mathcal{N}$ is drawn from some fixed probability distribution over the set of all deterministic depth-a d-networks.) □

LEMMA 4.9. *Let random* 0-1 *d-vector ϕ be drawn from an arbitrary probability distribution over $\Phi(d)$ and π be a random d-permutation drawn from $\Pi_R(d, a)$. Then the* 0-1 *d-vector ϕ' obtained by applying π to ϕ is a-random.*

Proof. The proof is straightforward. □

LEMMA 4.10. *Let $\mathcal{N}$ denote a depth-b hypercubic d-network, and for each i, $0 \leq i \leq b$, let $f^+(i)$ (resp., $f^-(i)$) denote the number of levels j of $\mathcal{N}$, $0 \leq j < i$, such that $\hookrightarrow_d$ (resp., $\leftrightarrow_d$) is applied in the permutation phase. Let $f(i) = f^+(i) - f^-(i)$, $0 \leq i \leq b$. For all a such that $0 \leq a < d$, $\mathcal{N}$ is a-partitionable if and only if* (i) *$0 \leq f(i) \leq a$, $0 \leq i < b$, and* (ii) *$f(b) = 0$.*

Proof. Let $g^+(i) = \max_{0 \leq j \leq i} f(j)$ and $g^-(i) = \min_{0 \leq j \leq i} f(j)$, $0 \leq i \leq b$. Note that $g^+(i) \geq 0$ and $g^-(i) \leq 0$, $0 \leq i \leq b$.

Let $A_{i,j}$ denote the set of level i output wires y of $\mathcal{N}$ such that there is a path to y from some input wire x in the jth input a-cube, $0 \leq i < b$, $0 \leq j < \mathrm{pow}(d - a)$.

It is straightforward to prove by induction on i that

$$A_{i,j} = \{y : j_{k-a} = y_{k-f(i)},\ \max\{a, g^+(i)\} \leq k < d - g^-(i)\}. \tag{1}$$

Thus,

$$|A_{i,j}| = \mathrm{pow}(\min\{d, \max\{a, g^+(i)\} - g^-(i)\}). \tag{2}$$

Note that if $|A_{i,j}| > \text{pow}(a)$ for some i and j, $0 \leq i < b$, $0 \leq j < \text{pow}(d-a)$, then $\mathcal{N}$ is not a-partitionable. It follows from equation (2) that $\mathcal{N}$ is not a-partitionable if $g^+(b) > a$ or $g^-(b) < 0$.

It remains to prove that if $g^-(b) = 0$ and $g^+(b) \leq a$, then $\mathcal{N}$ is a-partitionable if and only if $f(b) = 0$. Accordingly, assume that $g^-(b) = 0$ and $g^+(b) \leq a$. By equation (1), $A_{b,j}$ corresponds to the jth output subcube if and only of $f(b) = 0$, $0 \leq j < \text{pow}(d-a)$, completing the proof. □

LEMMA 4.11. *Let $\mathcal{N}$ be a random coin-tossing depth-a d-network, and p (resp., p') denote the probability that a particular output wire x receives a 0 when the input to $\mathcal{N}$ is a d-vector drawn from the probability distribution $\mathcal{D}$ (resp., $\mathcal{D}'$). If $\mathcal{D} \leq \mathcal{D}'$ then $p \geq p'$.*

Proof. If $\mathcal{N}$ is deterministic, the claim follows easily from consideration of the following "monotone" property of deterministic networks: When the value passed to a single input wire is changed from 0 to 1 (resp., from 1 to 0) no output changes from 1 to 0 (resp., 0 to 1).

If $\mathcal{N}$ is not deterministic, then it is given by some fixed probability distribution $\mathcal{D}''$ over the set of all deterministic depth-a d-networks. Since the claim holds for every deterministic network, we can prove that the claim holds for $\mathcal{N}$ by averaging over $\mathcal{D}''$. □

5. Analysis of the no-elimination tournament. Let us define a 0-1 *no-elimination (d,p)-tournament*, $d \geq 0$, as an execution of the depth-d shuffle-"+" d-network on a random 0-1 input d-vector drawn from the distribution $\Phi_B(d,p)$. In this section, we analyze the behavior of the 0-1 no-elimination tournament. Our analysis culminates with Lemma 5.16, which establishes that the 0-1 no-elimination tournament has a surprisingly strong ranking property. This ranking property is used to carry out the applications of subsequent sections. (It is noteworthy that the depth-d shuffle-"+" d-network is equivalent to Batcher's bitonic merge network. We have chosen not to adopt Batcher's terminology because we plan to expose a property of the network that is largely unrelated to merging.)

The central idea underlying the proof of Lemma 5.16 is that for almost all output wires x of a 0-1 no-elimination (d,p)-tournament, the probability that x receives a 0 (which is a function of p) exhibits sharp threshold behavior; there is a probability q (which depends on x) such that x is extremely unlikely (resp., likely) to receive a 0 whenever p is at least a bit smaller (resp., larger) than q. Thus, if we permute the output wires of the 0-1 no-elimination (d,p)-tournament according to the sorted order of their associated threshold probabilities, we will tend to produce an approximately sorted 0-1 output vector.

The proof of Lemma 5.16 is rather lengthy and is organized into a number of sections. In section 5.1 we define and analyze certain output probability polynomials. In section 5.2 we study the inverse functions associated with these output probability polynomials. In section 5.3 we provide a number of auxiliary definitions. In section 5.4 we present several technical lemmas. In section 5.5 we complete the proof of Lemma 5.16.

5.1. The output polynomials. In this section we analyze the probability that each wire in a 0-1 no-elimination (d,p)-tournament carries a 0. We define the output probability polynomials $\sigma_\alpha(p)$ and prove two basic lemmas concerning these polynomials.

LEMMA 5.1. *Let x_0 and x_1 denote the two intermediate wires associated with some gate in a 0-1 no-elimination (d,p)-tournament. Then the events E_0 = "x_0*

receives a 0" and E_1 = "x_1 receives a 0" are independent.

Proof. Assume without loss of generality that: (i) x_0 is intermediate wire $2 \cdot j$ on level i; and (ii) x_1 is intermediate wire $2 \cdot j+1$ on level i, $0 \leq i < d$, $0 \leq j < \text{pow}(d-1)$. Note that the index of every level 0 input wire with a path to x_0 has a 0 in bit position $d-i-1$. Similarly, the index of every level 0 input wire with a path to x_1 has a 1 in bit position $d-i-1$. Thus, wires x_0 and x_1 depend on disjoint subsets of the level 0 input wires, and the claim of the lemma follows. □

With each binary string α, we associate the function $\sigma_\alpha(p)$, defined inductively as follows:

(i) $\sigma_\epsilon(p) = p$;
(ii) $\sigma_{\alpha 0}(p) = 2 \cdot \sigma_\alpha(p) - \sigma_\alpha(p)^2$; and
(iii) $\sigma_{\alpha 1}(p) = \sigma_\alpha(p)^2$.

One may easily verify that for each binary string α, the following conditions hold: (i) $\sigma_\alpha(0) = 0$; (ii) $\sigma_\alpha(1) = 1$; (iii) $\sigma_\alpha(p)$ is monotonically increasing for p in $[0,1]$; (iv) $\sigma_\alpha(p)$ is a degree-pow($|\alpha|$) polynomial in p. Conditions (i), (ii), and (iii) imply that $\sigma_\alpha(p)$ is in $[0,1]$ for all p in $[0,1]$.

LEMMA 5.2. *Output wire j of a 0-1 no-elimination (d,p)-tournament receives a 0 with probability $\sigma_\alpha(p)$, where $\alpha = j_{d-1} \cdots j_0$.*

Proof. We prove instead the following stronger claim for all i and j such that $0 \leq i < d$, $0 \leq j < \text{pow}(d)$:

(i) Input wire j at level i receives a 0 with probability $\sigma_\alpha(p)$, where $\alpha = j_{i-1} \cdots j_0$.
(ii) Intermediate wire j at level i receives a 0 with probability $\sigma_\alpha(p)$, where $\alpha = j_i \cdots j_1$.
(iii) Output wire j at level i receives a 0 with probability $\sigma_\alpha(p)$, where $\alpha = j_i \cdots j_0$.

For $i = 0$, part (i) of the claim is immediate since $\sigma_\epsilon(p) = p$. Now let us assume that part (i) of the claim holds for some i, $0 \leq i < d$. Then part (ii) of the claim holds for i since the value received by input wire j is passed to intermediate wire $j_{d-2} \cdots j_0 j_{d-1}$. Similarly, it is easy to show that if part (iii) of the claim holds for some i, $0 \leq i < d-1$, then part (i) holds for $i+1$, since output wire j at level i is the same as input wire j at level $i+1$.

It remains only to prove that if part (ii) of the claim holds for some i, $0 \leq i < d$, then part (iii) holds for i. Accordingly, let x_0 and x_1 denote the pair of intermediate wires associated with some gate y at level i, assume that part (ii) of the claim holds for these wires, and that the associated d-bit indices of x_0 and x_1 are $\beta\alpha 0$ and $\beta\alpha 1$, respectively, where $|\alpha| = i$. Then

$$\Pr\{x_0 \text{ receives a } 0\} = \Pr\{x_1 \text{ receives a } 0\} = \sigma_\alpha(p),$$

and these probabilities are independent by Lemma 5.1. Hence, the "min" output of gate y (i.e., the output wire with index $\beta\alpha 0$ at level i) receives a 0 with probability

$$2 \cdot \sigma_\alpha(p) - \sigma_\alpha(p)^2 = \sigma_{\alpha 0}(p),$$

and the "max" output of gate y (i.e., the output wire with index $\beta\alpha 1$ at level i) receives a 0 with probability

$$\sigma_\alpha(p)^2 = \sigma_{\alpha 1}(p),$$

as required. □

Let α and β denote the binary sequences corresponding to the win-loss sequences `WLWLLWLL` and `LLLWWWWW` mentioned in section 1. We can easily calculate that $\sigma_\alpha(1/2) \approx$

0.796 and $\sigma_\beta(1/2) \approx 0.882$, suggesting that the player with record α should be rated above the player with record β.

LEMMA 5.3. *For all binary strings α and β, and all p in $[0,1]$,*

$$\sigma_{\beta\alpha}(p) = \sigma_\alpha(\sigma_\beta(p)).$$

Proof. For $\alpha = \epsilon$, the result is immediate since $\sigma_\epsilon(p) = p$. For $|\alpha| > 0$, we prove the result by induction on $|\alpha|$. For the base case, assume that $\alpha = x$ where x is either 0 or 1. Since $\sigma_0(p) = 2 \cdot p - p^2$ and $\sigma_1(p) = p^2$, we find that $\sigma_{\beta x}(p) = \sigma_x(\sigma_\beta(p))$, as required. Our induction hypothesis is that the claim holds for all α and β with $|\alpha| \leq i$, for some $i \geq 1$. For the induction step, we will prove that the claim holds for all α, β with $\alpha = \alpha' x$, x equal to 0 or 1, and $|\alpha'| = i$. The proof follows from three applications of the induction hypothesis, since

$$\sigma_{\beta\alpha}(p) = \sigma_{\beta\alpha' x}(p) = \sigma_x(\sigma_{\beta\alpha'}(p)) = \sigma_x(\sigma_{\alpha'}(\sigma_\beta(p))) = \sigma_{\alpha' x}(\sigma_\beta(p)) = \sigma_\alpha(\sigma_\beta(p)). \qquad \square$$

5.2. The inverses of the output polynomials. In order to better understand the behavior of the output polynomial σ_α, it will be useful to study its inverse function. In particular, for any binary string α, we define $\Gamma_\alpha(z)$ to be the function such that

$$\Gamma_\alpha(\sigma_\alpha(p)) = p$$

for all p in $[0,1]$. Unlike σ_α, Γ_α is not a polynomial for $|\alpha| \geq 1$. However, like σ_α, there is a simple inductive scheme for computing Γ_α. This is demonstrated by the following lemma.

LEMMA 5.4. *For all binary strings α, and all z in $[0,1]$,*

$$\begin{aligned} \Gamma_\epsilon(z) &= z, \\ \Gamma_{0\alpha}(z) &= 1 - \sqrt{1 - \Gamma_\alpha(z)}, \\ \Gamma_{1\alpha}(z) &= \sqrt{\Gamma_\alpha(z)}. \end{aligned}$$

Proof. Since $\sigma_\epsilon(p) = p$ for all p in $[0,1]$, σ_ϵ is the identity function, and thus Γ_ϵ is also the identity function. Hence $\Gamma_\epsilon(z) = z$ for all z in $[0,1]$.

By Lemma 5.3, we have

$$\begin{aligned} \sigma_{0\alpha}(p) &= \sigma_\alpha(\sigma_0(p)) \\ &= \sigma_\alpha(2 \cdot p - p^2) \end{aligned}$$

for all p in $[0,1]$. Setting $p = \Gamma_{0\alpha}(z)$, we find that

$$\begin{aligned} \sigma_\alpha(2 \cdot \Gamma_{0\alpha}(z) - \Gamma_{0\alpha}(z)^2) &= \sigma_{0\alpha}(\Gamma_{0\alpha}(z)) \\ &= z \\ &= \sigma_\alpha(\Gamma_\alpha(z)). \end{aligned}$$

Since σ_α is a monotonically increasing function, we have

$$2 \cdot \Gamma_{0\alpha}(z) - \Gamma_{0\alpha}(z)^2 = \Gamma_\alpha(z).$$

Solving for $\Gamma_{0\alpha}(z)$, we obtain

$$\Gamma_{0\alpha}(z) = 1 - \sqrt{1 - \Gamma_\alpha(z)},$$

as desired.

The proof that $\Gamma_{1\alpha}(z) = \sqrt{\Gamma_\alpha(z)}$ proceeds in a similar fashion. By Lemma 5.3, we have

$$\begin{aligned}\sigma_{1\alpha}(p) &= \sigma_\alpha(\sigma_1(p)) \\ &= \sigma_\alpha(p^2)\end{aligned}$$

for all p in $[0,1]$. Setting $p = \Gamma_{1\alpha}(z)$, we find that

$$\begin{aligned}\sigma_\alpha(\Gamma_{1\alpha}(z)^2) &= \sigma_{1\alpha}(\Gamma_{1\alpha}(z)) \\ &= z \\ &= \sigma_\alpha(\Gamma_\alpha(z)).\end{aligned}$$

Since σ_α is a monotonically increasing function, we have

$$\Gamma_{1\alpha}(z)^2 = \Gamma_\alpha(z)$$

and thus

$$\Gamma_{1\alpha}(z) = \sqrt{\Gamma_\alpha(z)},$$

as desired. □

Let α and β denote the binary sequences corresponding to the win-loss sequences `WLWLLWLL` and `LLLWWWWW` mentioned in section 1. We can easily calculate that $\Gamma_\alpha(1/2) \approx 0.437$ and $\Gamma_\beta(1/2) \approx 0.381$, suggesting that the player with record α should be rated above the player with record β.

Note that $\Gamma_\alpha(0) = 0$ and $\Gamma_\alpha(1) = 1$ for all binary strings α. The following lemma is analogous to Lemma 5.3.

LEMMA 5.5. *For all binary strings α and β, and all z in $[0,1]$,*

$$\Gamma_{\beta\alpha}(z) = \Gamma_\beta(\Gamma_\alpha(z)).$$

Proof. The proof is similar to that of Lemma 5.3. □

5.3. Auxiliary definitions. In this section, we state a number of definitions related to the analysis of the no-elimination tournament. These definitions are used primarily in sections 5.4 and 5.5, but also appear in subsequent sections.

For all $x < y$ in $[0,1]$, $\lambda \geq 1$, and $d \geq 0$, let

$$\Delta(x,y) = \lg \frac{y \cdot (1-x)}{(1-y) \cdot x}, \tag{3}$$

$$h_\alpha(x,y) = \frac{\Delta(\Gamma_\alpha(x), \Gamma_\alpha(y))}{\Delta(x,y)}, \tag{4}$$

$$H_\lambda(x,y,d) = \sum_{\alpha:|\alpha|=d} h_\alpha(x,y)^\lambda, \tag{5}$$

$$\eta(\lambda) = \sup_{0<x<y<1} \left\{ h_0(x,y)^\lambda + h_1(x,y)^\lambda \right\}, \tag{6}$$

$$\gamma_c = \inf_{\lambda \geq 1} \frac{\lg \eta(\lambda) + \lambda}{\lambda + 1} \approx 0.822, \tag{7}$$

$$\mu_c = \sup_{\lambda \geq 1} \frac{1 - \lg \eta(\lambda)}{\lambda} = \lg(4 - 2 \cdot \sqrt{2}) \approx 0.228, \tag{8}$$

$$\upsilon_c = -2 \cdot \lg \mu_c = -2 \cdot \lg \lg(4 - 2 \cdot \sqrt{2}) \approx 4.260. \tag{9}$$

Informally, we think of $\Delta(x, y)$ as a measure of the "distance" between x and y for $x < y$ in $[0, 1]$. The function $h_\alpha(x, y)$ may then be viewed as the fractional decrease in the distance between x and y that results from applying Γ_α to both x and y. In section 5.4, we use an inductive potential argument, with potential function $H(x, y, d)$, to show that $h_\alpha(x, y)$ is very small for most α. The function $\eta(\lambda)$ arises in the process of bounding the size of the potential function. The constants γ_c and μ_c are related to the notion of an admissible triple, which we define below. (Note that the constant γ_c and μ_c appear in the statements of Lemmas 5.6 and 5.7, respectively.) The constant υ_c appears in the exponent of the depth bound of section 9.

Setting $\lambda = 3$, we can use Lemma 5.11 and elementary calculus to show that

$$\eta(3) = \frac{10 + 7 \cdot \sqrt{2}}{16},$$

which is attainable for $z = 1/2$. This implies $\gamma_c \leq [\lg(10 + 7 \cdot \sqrt{2}) - 1]/4 \approx 0.829$ and $\mu_c \geq [5 - \lg(10 + 7 \cdot \sqrt{2})]/3 = \lg(4 - 2 \cdot \sqrt{2}) \approx 0.228$. Using numerical calculations, it can be shown that $\gamma_c \approx 0.822$, which is attained for $\lambda \approx 3.609$ and $\eta(\lambda) \approx 1.133$. On the other hand, the supremum of $(1 - \lg \eta(\lambda))/\lambda$, $\lambda \geq 1$, is actually achieved for $\lambda = 3$, so $\mu_c = \lg(4 - 2 \cdot \sqrt{2})$.

DEFINITION 5.1. *A triple (γ, ε, d), where $0 < \gamma < 1$, $0 \leq \varepsilon < 1/2$, and $d \geq 0$, is defined to be* admissible *if and only if*

$$\gamma \cdot d \cdot (\lambda + 1) \geq d \cdot \lg \eta(\lambda) + \lambda \cdot [d + \lg \lg(1/\varepsilon) + 2 - \lg(1 - 2 \cdot \varepsilon)]$$

for some $\lambda \geq 1$.

Definition 5.1 is somewhat messy to apply directly. The following pair of technical lemmas characterize two classes of admissible triples that arise in our applications.

LEMMA 5.6. *For each function $\varepsilon(d) = \text{pow}(-\text{pow}(O(d)))$, there is a function*

$$f(d) = O(\lg \lg(1/\varepsilon(d)))/d$$

such that $(\gamma, \varepsilon(d), d)$ is an admissible triple for all $d \geq 0$ and $\gamma_c + f(d) \leq \gamma < 1$.

Proof. This follows from routine calculations involving Definition 5.1 and equation (7). □

LEMMA 5.7. *For each function $\varepsilon(d) = \text{pow}(-\text{pow}(\mu \cdot d))$, where $\mu(d) = \mu_c - \frac{1}{f(d)}$ with $f(d) = \omega(1)$ and $f(d) = o(d)$, there is a function $g(d) = o(1)$ such that*

$$\left(1 - \frac{3 - g(d)}{4 \cdot f(d)}, \varepsilon(d), d\right)$$

is an admissible triple for all $d \geq 0$.

Proof. This follows from routine calculations involving Definition 5.1 and equation (8). □

5.4. Several technical lemmas. In this section, we prove a number of technical lemmas that are used only in section 5.5 of the paper. Lemma 5.8 shows that the notion of "distance" associated with the function $\Delta(x, y)$ is always at least twice the difference $y - x$. Lemma 5.9 provides a useful method for recursively rewriting expressions of the form $h_{\beta\alpha}(x, y)$. Lemma 5.10 gives a maximization result that is used within Lemma 5.11 to obtain a simplified definition of $\eta(\lambda)$. Lemma 5.12 proves that the potential function $H(x, y, d)$ is bounded from above by $\eta(\lambda)^d$. Lemma 5.13

shows that for certain small values of ε, the difference $\Gamma_\alpha(1-\varepsilon) - \Gamma_\alpha(\varepsilon)$ is small for most binary strings α.

LEMMA 5.8. *For all $x < y$ in $[0,1]$,*

$$y - x \le \Delta(x,y)/2.$$

Proof. Define

$$\rho(z) = \frac{1}{4} \cdot \ln \frac{z}{1-z} - z$$

for z in $[0,1]$. Since

$$\frac{d\rho(z)}{dz} = \frac{1}{4} \cdot \left(\frac{1}{z} + \frac{1}{1-z} \right) - 1 \ge 0$$

for z in $[0,1]$, we know that $\rho(z)$ is a nondecreasing function of z. Hence,

$$\begin{aligned} \frac{\Delta(x,y)}{4 \cdot \lg e} - (y-x) &= \frac{\lg \frac{y \cdot (1-x)}{(1-y) \cdot x}}{4 \cdot \lg e} - y + x \\ &= \rho(y) - \rho(x) \\ &\ge 0, \end{aligned}$$

and thus

$$y - x \le \frac{\Delta(x,y)}{4 \lg e} \le \Delta(x,y)/2. \qquad \square$$

LEMMA 5.9. *For all $x < y$ in $[0,1]$, we have* (i) $h_\epsilon(x,y) = 1$, *and* (ii) *for all binary strings α and β,*

$$h_{\beta\alpha}(x,y) = h_\beta(\Gamma_\alpha(x), \Gamma_\alpha(y)) \cdot h_\alpha(x,y).$$

Proof. Since Γ_ϵ is the identity function, $h_\epsilon(x,y) = 1$ for all $x < y$ in $[0,1]$. We prove the second part of the lemma by observing that

$$\begin{aligned} h_{\beta\alpha}(x,y) &= \frac{\Delta(\Gamma_{\beta\alpha}(x), \Gamma_{\beta\alpha}(y))}{\Delta(x,y)} \\ &= \frac{\Delta(\Gamma_{\beta\alpha}(x), \Gamma_{\beta\alpha}(y))}{\Delta(\Gamma_\alpha(x), \Gamma_\alpha(y))} \cdot \frac{\Delta(\Gamma_\alpha(x), \Gamma_\alpha(y))}{\Delta(x,y)} \\ &= \frac{\Delta(\Gamma_\beta(\Gamma_\alpha(x)), \Gamma_\beta(\Gamma_\alpha(y)))}{\Delta(\Gamma_\alpha(x)\Gamma_\alpha(y))} \cdot h_\alpha(x,y) \\ &= h_\beta(\Gamma_\alpha(x), \Gamma_\alpha(y)) \cdot h_\alpha(x,y), \end{aligned}$$

where the second last equation follows from Lemma 5.5. $\square$

LEMMA 5.10. *Let f_0, f_1, and f denote strictly increasing and continuously differentiable functions on $(0,1)$, and set*

$$g(x,y,\lambda) = \left(\frac{f_0(y) - f_0(x)}{f(y) - f(x)} \right)^\lambda + \left(\frac{f_1(y) - f_1(x)}{f(y) - f(x)} \right)^\lambda$$

for $0 < x \le y < 1$ and $\lambda \ge 1$. Then for all $x \le y$ in $(0,1)$,

$$g(x,y,\lambda) \le \sup_{z\in(0,1)} g(z,z,\lambda).$$

Proof. Note that because f_0, f_1, and f are strictly increasing and differentiable, l'Hôpital's rule implies that $g(x,y,\lambda)$ is well defined even if $x = y$.

Given any $x < y$ in $(0,1)$, we prove below that there exists a p such that $x < p < y$ and

$$\max\{g(x,p,\lambda), g(p,y,\lambda)\} \ge g(x,y,\lambda).$$

This is sufficient to prove that the maximum of $g(x,y,\lambda)$ occurs for $x \sim y$.

Choose p so that

$$f(p) = \frac{f(x)+f(y)}{2}.$$

We can always find such a p between x and y since f is a continuous function. Then set

$$\begin{aligned} u_0 &= f_0(p) - f_0(x), \\ u_1 &= f_0(y) - f_0(p), \\ v_0 &= f_1(p) - f_1(x), \\ v_1 &= f_1(y) - f_1(p), \\ w &= f(p) - f(x) \\ &= f(y) - f(p). \end{aligned}$$

Note that u_0, u_1, v_0, v_1, and w are all strictly positive since f_0, f_1, and f are strictly increasing.

By definition,

$$\begin{aligned} g(x,p,\lambda) &= \left(\frac{u_0}{w}\right)^\lambda + \left(\frac{v_0}{w}\right)^\lambda, \\ g(p,y,\lambda) &= \left(\frac{u_1}{w}\right)^\lambda + \left(\frac{v_1}{w}\right)^\lambda, \\ g(x,y,\lambda) &= \left(\frac{u_0+u_1}{2\cdot w}\right)^\lambda + \left(\frac{v_0+v_1}{2\cdot w}\right)^\lambda. \end{aligned}$$

For $\lambda \ge 1$, the function z^λ is convex, and thus

$$\frac{z_0^\lambda + z_1^\lambda}{2} \ge \left(\frac{z_0+z_1}{2}\right)^\lambda$$

for all z_0 and z_1. Hence,

$$\begin{aligned} \left(\frac{u_0}{w}\right)^\lambda + \left(\frac{u_1}{w}\right)^\lambda &\ge 2\cdot\left(\frac{u_0+u_1}{2\cdot w}\right)^\lambda, \\ \left(\frac{v_0}{w}\right)^\lambda + \left(\frac{v_1}{w}\right)^\lambda &\ge 2\cdot\left(\frac{v_0+v_1}{2\cdot w}\right)^\lambda. \end{aligned}$$

Summing the preceding pair of inequalities, we find that

$$g(x,p,\lambda) + g(p,y,\lambda) \geq 2 \cdot g(x,y,\lambda),$$

and hence $\max\{g(x,p,\lambda), g(p,y,\lambda)\} \geq g(x,y,\lambda)$, as desired. □

LEMMA 5.11. *For all* $\lambda \geq 1$,

$$\eta(\lambda) = \sup_{0 \leq z \leq 1} \left[\left(\frac{1+\sqrt{z}}{2} \right)^{\lambda} + \left(\frac{1+\sqrt{1-z}}{2} \right)^{\lambda} \right].$$

Proof. We have

$$\begin{aligned}
&h_0(x,y)^{\lambda} + h_1(x,y)^{\lambda} \\
&= \left(\frac{\Delta(\Gamma_0(x), \Gamma_0(y))}{\Delta(x,y)} \right)^{\lambda} + \left(\frac{\Delta(\Gamma_1(x), \Gamma_1(y))}{\Delta(x,y)} \right)^{\lambda} \\
&= \left(\frac{\lg \frac{\Gamma_0(y) \cdot (1-\Gamma_0(x))}{(1-\Gamma_0(y)) \cdot \Gamma_0(x)}}{\lg \frac{y \cdot (1-x)}{(1-y) \cdot x}} \right)^{\lambda} + \left(\frac{\lg \frac{\Gamma_1(y) \cdot (1-\Gamma_1(x))}{(1-\Gamma_1(y)) \cdot \Gamma_1(x)}}{\lg \frac{y \cdot (1-x)}{(1-y) \cdot x}} \right)^{\lambda} \\
&= \left(\frac{\lg \frac{\Gamma_0(y)}{1-\Gamma_0(y)} - \lg \frac{\Gamma_0(x)}{1-\Gamma_0(x)}}{\lg \frac{y}{1-y} - \lg \frac{x}{1-x}} \right)^{\lambda} + \left(\frac{\lg \frac{\Gamma_1(y)}{1-\Gamma_1(y)} - \lg \frac{\Gamma_1(x)}{1-\Gamma_1(x)}}{\lg \frac{y}{1-y} - \lg \frac{x}{1-x}} \right)^{\lambda} \\
&= \left(\frac{f_0(y) - f_0(x)}{f(y) - f(x)} \right)^{\lambda} + \left(\frac{f_1(y) - f_1(x)}{f(y) - f(x)} \right)^{\lambda},
\end{aligned}$$

where

$$\begin{aligned}
f_0(z) &= \lg \frac{\Gamma_0(z)}{1 - \Gamma_0(z)} \\
&= \lg \frac{1 - \sqrt{1-z}}{\sqrt{1-z}}, \\
f_1(z) &= \lg \frac{\Gamma_1(z)}{1 - \Gamma_1(z)} \\
&= \lg \frac{\sqrt{z}}{1 - \sqrt{z}}, \\
f(z) &= \lg \frac{z}{1-z}.
\end{aligned}$$

It is easily verified that $f_0(z)$, $f_1(z)$, and $f(z)$ are strictly increasing and continuously differentiable in $(0,1)$. By Lemma 5.10, this means that the supremum of $h_0(x,y)^{\lambda} + h_1(x,y)^{\lambda}$ occurs for $x \sim y$. Using l'Hôpital's rule and elementary calculus, we can show that

$$\begin{aligned}
\lim_{\varepsilon \to 0} h_0((1-\varepsilon) \cdot y, y) &= \frac{df_0(y)/dy}{df(y)/dy} \\
&= \frac{1 + \sqrt{1-y}}{2}.
\end{aligned}$$

Reasoning in a similar fashion, we can also show that

$$\lim_{\varepsilon \to 0} h_1((1-\varepsilon) \cdot y, y) = \frac{df_1(y)/dy}{df(y)/dy}$$

$$= \frac{1+\sqrt{y}}{2}.$$

The proof of the lemma now follows from the definition of $\eta(\lambda)$. □

LEMMA 5.12. *For all $x < y$ in $[0,1]$, $\lambda \geq 1$, and $d \geq 0$, we have*

$$H_\lambda(x,y,d) \leq \eta(\lambda)^d.$$

Proof. For $d = 0$, the result is immediate since $H_\lambda(x,y,0) = 1$. For $d > 0$, Lemma 5.9 implies that

$$\begin{aligned} H_\lambda(x,y,d) &= \sum_{\alpha:|\alpha|=d-1} \left(h_{0\alpha}(x,y)^\lambda + h_{1\alpha}(x,y)^\lambda\right) \\ &= \sum_{\alpha:|\alpha|=d-1} \left(h_0(\Gamma_\alpha(x),\Gamma_\alpha(y))^\lambda + h_1(\Gamma_\alpha(x),\Gamma_\alpha(y))^\lambda\right) \cdot h_\alpha(x,y)^\lambda \\ &\leq \sum_{\alpha:|\alpha|=d-1} \eta(\lambda) \cdot h_\alpha(x,y)^\lambda \\ &= \eta(\lambda) \cdot H_\lambda(x,y,d-1). \end{aligned}$$

Hence, $H_\lambda(x,y,d) \leq \eta(\lambda)^d$, as required. □

LEMMA 5.13. *For any admissible triple (γ,ε,d), there are at most $\mathrm{pow}(\gamma \cdot d)$ length-d binary strings α such that*

$$\Gamma_\alpha(1-\varepsilon) - \Gamma_\alpha(\varepsilon) > (1 - 2\cdot\varepsilon) \cdot \mathrm{pow}((\gamma-1)\cdot d)/4.$$

Proof. By Lemma 5.8, the definition of h_α, and the definition of Δ, we have

$$\begin{aligned} \Gamma_\alpha(1-\varepsilon) - \Gamma_\alpha(\varepsilon) &\leq \Delta(\Gamma_\alpha(\varepsilon),\Gamma_\alpha(1-\varepsilon))/2 \\ &= h_\alpha(\varepsilon,1-\varepsilon)\cdot\Delta(\varepsilon,1-\varepsilon)/2 \\ &\leq h_\alpha(\varepsilon,1-\varepsilon)\cdot\lg(1/\varepsilon). \end{aligned}$$

Hence, it is sufficient to prove that at most $\mathrm{pow}(\gamma\cdot d)$ length-d binary strings α satisfy

$$h_\alpha(\varepsilon,1-\varepsilon) > \frac{(1-2\cdot\varepsilon)\cdot\mathrm{pow}((\gamma-1)\cdot d)}{4\cdot\lg(1/\varepsilon)}.$$

Suppose the latter claim were false. Then

$$\begin{aligned} H_\lambda(\varepsilon,1-\varepsilon,d) &> \mathrm{pow}(\gamma\cdot d)\cdot\left[\frac{(1-2\cdot\varepsilon)\cdot\mathrm{pow}((\gamma-1)\cdot d)}{4\cdot\lg(1/\varepsilon)}\right]^\lambda \\ &= \mathrm{pow}(\gamma\cdot d\cdot(\lambda+1) - \lambda\cdot[d + \lg\lg(1/\varepsilon) + 2 - \lg(1-2\cdot\varepsilon)]) \\ &\geq \eta(\lambda)^d, \end{aligned}$$

which contradicts Lemma 5.12. □

5.5. The no-elimination tournament theorem. In this section, we complete the proof of Lemma 5.16.

LEMMA 5.14. *For any admissible triple (γ,ε,d), there exists a set A of at least $\mathrm{pow}(d) - \mathrm{pow}(\gamma\cdot d)$ output wires of the depth-d shuffle-"+" d-network, and a fixed permutation π of A, such that for each p the set A can be partitioned into three sets B, A^-, and A^+ where:*

(i) *the set of output wires B is mapped to a contiguous interval by the permutation π,*
(ii) $|B| < \mathrm{pow}(\gamma \cdot d)$,
(iii) A^- *(resp., A^+) is the set of all output wires in $A \setminus B$ mapped to positions lower (resp., higher) than B by π, and*
(iv) *after execution of a* 0-1 *no-elimination (d,p)-tournament, each output wire in A^- (resp., A^+) receives a* 1 *(resp.,* 0*) with probability less than ε.*

Proof. Let (γ, ε, d) denote a given admissible triple, and choose A to be the set (guaranteed to exist by Lemma 5.13) of at least $\mathrm{pow}(d) - \mathrm{pow}(\gamma \cdot d)$ output wires indexed by length-d binary strings α such that

$$\Gamma_\alpha(1-\varepsilon) - \Gamma_\alpha(\varepsilon) \le \delta,$$

where $\delta = (1 - 2 \cdot \varepsilon) \cdot \mathrm{pow}((\gamma - 1) \cdot d)/4$. (Note that $0 < \delta < 1/4$ since (γ, ε, d) is an admissible triple.) We remark that, using Lemma 5.4, $\Gamma_\alpha(z)$ can be computed in $O(|\alpha|)$ arithmetic operations (counting square root as a single operation) for any z in $[0,1]$. Hence, the set A can be computed in $O(d \cdot \mathrm{pow}(d))$ operations. (This may be viewed as a relatively efficient time bound since, for example, it is linear in the size of the depth-d shuffle-"+" d-network.)

In fact, we can compute an appropriate permutation π within the same asymptotic time bound: We set π to the permutation of set A that sorts the $\Gamma_\alpha(\varepsilon)$ values in ascending order. Ties may be broken arbitrarily. It remains to prove that our choice of A and π satisfies the requirements of the lemma.

Let $p^- = \max\{0, p - \delta\}$ and $p^+ = \min\{1, p + \delta\}$. (Recall that p is the 0-1 no-elimination tournament input parameter.) Let B denote the set of binary strings α in A for which $\Gamma_\alpha(\varepsilon)$ is contained in $[p^-, p]$. Because the σ_α's are monotonically increasing, and using linearity of expectation, we have

$$\begin{aligned}
\sum_{\alpha:|\alpha|=d} |\sigma_\alpha(p^+) - \sigma_\alpha(p^-)| &= \sum_{\alpha:|\alpha|=d} \big(\sigma_\alpha(p^+) - \sigma_\alpha(p^-)\big) \\
&= \left(\sum_{\alpha:|\alpha|=d} \sigma_\alpha(p^+)\right) - \left(\sum_{\alpha:|\alpha|=d} \sigma_\alpha(p^-)\right) \\
&= (p^+ - p^-) \cdot \mathrm{pow}(d) \\
&\le 2 \cdot \delta \cdot \mathrm{pow}(d).
\end{aligned}$$

For each α in B we have $\sigma_\alpha(p^-) \le \varepsilon$. (If $p^- = 0$ then $\sigma_\alpha(p^-) = \sigma_\alpha(0) = 0$, and if $p^- = p - \delta$ then $\sigma_\alpha(p^-) \le \sigma_\alpha(\Gamma_\alpha(\varepsilon)) = \varepsilon$.) Furthermore, for each α in B we have $\sigma_\alpha(p^+) \ge 1 - \varepsilon$. (If $p^+ = 1$ then $\sigma_\alpha(p^+) = \sigma_\alpha(1) = 1$, and if $p^+ = p + \delta$ then $\sigma_\alpha(p^+) \ge \sigma_\alpha(\Gamma_\alpha(1-\varepsilon)) = 1 - \varepsilon$.) Hence,

$$|\sigma_\alpha(p^+) - \sigma_\alpha(p^-)| \ge 1 - 2 \cdot \varepsilon.$$

The preceding inequalities imply that

$$\begin{aligned}
|B| &\le 2 \cdot \delta \cdot \mathrm{pow}(d)/(1 - 2 \cdot \varepsilon) \\
&< \mathrm{pow}(\gamma \cdot d).
\end{aligned}$$

Note that the set of binary strings B satisfies conditions (i) and (ii) of the lemma. For the given choice of B, define sets A^- and A^+ to satisfy condition (iii). It remains only to address condition (iv).

Let α denote the binary string associated with an arbitrary output in A^-. Thus, $\Gamma_\alpha(\varepsilon) < p^-$, which implies $\Gamma_\alpha(1-\varepsilon) < p$. Hence, $\sigma_\alpha(p) > 1-\varepsilon$. (The probability that output α receives a 1 is less than ε.)

Similarly, let α denote the binary string associated with some output in A^+. Thus, $\Gamma_\alpha(\varepsilon) > p$ and hence $\sigma_\alpha(p) < \varepsilon$. (The probability that output α receives a 0 is less than ε.) □

LEMMA 5.15. *Let d, k, n, and p be such that $d \geq 0$, $n = \mathrm{pow}(d)$, $0 \leq k < n$, and $p = k/n$. Let $\mathcal{N}$ denote an arbitrary d-network, and assume that output wire x of $\mathcal{N}$ receives a 0 (resp., 1) with probability $q \leq \varepsilon$ when the input to $\mathcal{N}$ is drawn from $\Phi_B(d,p)$. Further assume that output wire x receives a 0 (resp., 1) with probability q_i when the input is drawn from $\Phi_R(d,i)$, $0 \leq i < n$. Then $q_k \leq 2 \cdot \varepsilon$.*

Proof. Informally, the lemma states that network $\mathcal{N}$ behaves similarly on inputs drawn from $\Phi_B(d,p)$ and $\Phi_R(d,k)$. Note that

$$\begin{aligned} q &= \sum_{0 \leq i < n} \binom{n}{i} p^i (1-p)^{n-i} q_i \\ &\geq \sum_{k \leq i < n} \binom{n}{i} p^i (1-p)^{n-i} q_i \\ &\geq q_k \sum_{k \leq i < n} \binom{n}{i} p^i (1-p)^{n-i} \\ &\geq q_k/2, \end{aligned}$$

where the last two inequalities follow from Lemma 4.11 and Theorem B.1, respectively. □

LEMMA 5.16. *For any admissible triple (γ, ε, d), we have*

$$\mathit{Most}^h_D(d, d, \lfloor \gamma \cdot d \rfloor, O(\mathrm{pow}(d) \cdot \varepsilon)) \leq d.$$

Proof. Let $\mathcal{N}$ denote the depth-d shuffle-"+" d-network. It follows from Lemmas 5.14 and 5.15 that $\mathcal{N}$ belongs to $\mathit{Most}^h_N(d, d, \lfloor \gamma \cdot d \rfloor, O(\mathrm{pow}(d) \cdot \varepsilon))$, justifying the claimed inequality.

(We have opted to omit the detailed proof of the above claim, which is straightforward but tedious. An outline of the proof is as follows. First, we rewrite the proof obligation by successively expanding the definitions of $\mathit{Most}^h_N(d, d, \lfloor \gamma \cdot d \rfloor, O(\mathrm{pow}(d) \cdot \varepsilon))$, $\lfloor \gamma \cdot d \rfloor$-mostly-sorted, and $\lfloor \gamma \cdot d \rfloor$-sorted. Having done this, we arrive at a new proof obligation in which the input to network $\mathcal{N}$ is assumed to be drawn from $\Phi_R(d,k)$, $0 \leq k < \mathrm{pow}(d)$. Furthermore, the new proof obligation is such that it would follow immediately from Lemma 5.14 if only the input were assumed to be drawn from $\Phi_B(d,p)$ instead of $\Phi_R(d,k)$. Lemma 5.15 is then used to complete the proof by relating the behavior of network $\mathcal{N}$ on inputs drawn from $\Phi_B(d,p)$ to its behavior on inputs drawn from $\Phi_R(d,k)$.) □

LEMMA 5.17. *For any admissible triple (γ, ε, a) and all d, $0 \leq a \leq d$, we have*

$$\begin{aligned} &\mathit{Most}_D(d, a, \lfloor \gamma \cdot a \rfloor, O(\mathrm{pow}(a) \cdot \varepsilon)) \leq a, \\ &\mathit{Most}^h_D(d, a, \lfloor \gamma \cdot a \rfloor, O(\mathrm{pow}(a) \cdot \varepsilon)) = O(a). \end{aligned}$$

Proof. It follows easily from Lemma 5.16 that the (d,a)-network consisting of the d-permutation $\hookrightarrow^a_d$ followed by the depth-a shuffle-"+" d-network has the desired properties. The additional depth required to implement the d-permutation $\hookrightarrow^a_d$ is

$\min\{a, d-a\} \leq a$ for a hypercubic construction and 0 for a nonhypercubic construction. □

Lemma 5.16 formalizes a central claim of the paper, namely, that the 0-1 no-elimination tournament has a surprisingly strong ranking property. Though stated in the 0-1 domain, Lemma 5.16 can easily be interpreted in the permutation domain. Such an interpretation is provided by the following theorem, which is stated without proof. (We remark that the extra factor of $\text{pow}(d)$ in the error bound arises because $|\Phi_\pi| = \text{pow}(d)+1$ for any d-permutation π. Also, Lemma 5.17, and not Theorem 5.1, is used to derive the results of subsequent sections.)

THEOREM 5.1. *Let (γ, ε, d) be an admissible triple, and define set A and permutation π as in the proof of Lemma* 5.14. *Let a random d-permutation drawn from $\Pi_R(d)$ be input to a depth-d shuffle-"+" d-network, and let π' denote the permutation induced on the output wires of A. Then the permutation obtained by applying π to π' is sorted to within* $\text{pow}(\gamma \cdot d)$ *positions with probability at least* $1 - O(\text{pow}(2 \cdot d) \cdot \varepsilon)$. □

Finally, we remark that the factors of $\text{pow}(d)$, $\text{pow}(a)$, and $\text{pow}(2 \cdot d)$ appearing in the error bounds associated with Lemmas 5.16 and 5.17 and Theorem 5.1 are not best possible. Lowering these factors would require a much more careful analysis, however, and from a theoretical point of view, would yield essentially no improvement in the results of subsequent sections. (Our applications make use of Lemma 5.17 with ε set far smaller than $\text{pow}(-c \cdot d)$ for any constant $c > 0$.) Of course, from a practical standpoint, it would be interesting to pin down the error bounds more accurately. For sufficiently small values of d, we remark that a computer program can be used to obtain very accurate error estimates.

6. A small-constant-factor network that sorts most inputs. Given Lemma 5.16, it is now a relatively simple task to design a logarithmic-depth network that sorts a random 0-1 input vector with high probability. The transformation consists of two basic components informally outlined below:

1. A procedure for augmenting the network of Lemma 5.16, which guarantees that all but a small fixed subset of the outputs are approximately sorted on a random 0-1 input vector with high probability, to obtain a network which guarantees that the entire output vector is approximately sorted on a random 0-1 input vector with high probability.
2. Recursive application of the network obtained from the previous step, with occasional merge operations in order to correct for items that fall into the wrong recursive subproblem due to boundary effects.

If the network of Lemma 5.16 worked on all 0-1 input vectors, and if we didn't care about constant factors, then it would be straightforward to devise a logarithmic-depth sorting network using the approach described above. Since we do care about constant factors and have to worry about probabilities, however, our solution will be somewhat more involved, and the proof will be somewhat more tedious. Nevertheless, we will still follow the basic approach outlined above in order to establish Lemma 6.1, the main technical result of this section.

LEMMA 6.1. *For all a and d such that $0 \leq a \leq d$, and each function $\varepsilon(d) = \text{pow}(-\text{pow}(o(d)))$, there is a function $f(d) = o(1)$ such that*

$$Sort_D(d, a, \varepsilon(a)) \leq \frac{2 - \gamma_c^2 + f(a)}{1 - \gamma_c} \cdot a.$$

Furthermore, there is a deterministic d-network that achieves this bound.

Note that $(2-\gamma_c^2)/(1-\gamma_c) \approx 7.44$. The following theorem provides an interpretation of Lemma 6.1 in the permutation domain.

THEOREM 6.1. *For each function $\varepsilon(d) = \mathrm{pow}(-\mathrm{pow}(o(d)))$, there is a function $f(d) = o(1)$ and a deterministic d-network of depth*

$$\frac{2-\gamma_c^2+f(d)}{1-\gamma_c} \cdot d$$

that sorts a random d-permutation drawn from $\Pi_R(d)$ with probability at least $1-\varepsilon(d)$.

Proof. Let $n = \mathrm{pow}(d)$. If π is a random d-permutation drawn from $\Pi_R(d)$, then ϕ_π^k is drawn at random from $\Phi_R(d,k)$, $0 \le k \le n$. By Lemma 6.1 (with $a = d$), there is a deterministic d-network $\mathcal{N}$ of the desired depth that sorts a random 0-1 d-vector drawn from $\Phi_R(d,k)$, $0 \le k \le n$, with probability at least $1-\varepsilon(d)$ for each function $\varepsilon(d) = \mathrm{pow}(-\mathrm{pow}(g(d)))/(n+1)$ with $g(d) = o(d)$. By Lemma 4.2, d-network $\mathcal{N}$ sorts d-permutation π if and only if it sorts the $n+1$ 0-1 d-vectors in Φ_π. This occurs with probability at least $1-(n+1)\cdot\varepsilon(d) = 1-\mathrm{pow}(-\mathrm{pow}(g(d)))$, as required. □

LEMMA 6.2. *For all a, b, and d such that $0 \le b \le a \le d$, and all ε and ε' in $[0,1]$, we have*

$$\mathit{Sort}_D(d, a, \varepsilon + 2\cdot\varepsilon') \le \mathit{Sort}_D(d,a,b,\varepsilon) + \mathit{Sort}_D(d,b,\varepsilon') + 2\cdot\mathit{Merge}_D(d,b,1).$$

Proof. We may assume that $a > b$, since the claim is trivial otherwise. We argue that a (d,a)-network in $\mathit{Sort}_N(d, a, \varepsilon + 2\cdot\varepsilon')$ can be constructed by composing the following: (a) any (d,a)-network in $\mathit{Sort}_N(d,a,b,\varepsilon)$; (b) a random d-permutation drawn from $\Pi_R(d,b)$; (c) any (d,a)-network in $\mathit{Sort}_N(d,b,\varepsilon')$; (d) any $(d,b+1)$-network in $\mathit{Merge}_N(d,b,1)$; (e) the d-permutation π in $\Pi(d,a)$ that maps wire i to wire $(i+\mathrm{pow}(b)) \bmod \mathrm{pow}(a)$, $0 \le i < \mathrm{pow}(a)$, within each a-cube; (f) any $(d,b+1)$-network in $\mathit{Merge}_N(d,b,1)$; (g) the d-permutation π^{-1}; and (h) the d-permutation in $\Pi(d,a)$ that exchanges the lowest and highest b-cubes within each a-cube. (Note that this construction does indeed give a (d,a)-network.)

We may assume that the input is an a-random 0-1 d-vector. Consider an arbitrary a-cube A. After stage (a) of the construction, A is b-sorted with probability at least $1-\varepsilon$. The output of stage (b) is b-random by Lemma 4.9. Hence, each b-cube of A is sorted with probability at least $1-\varepsilon'$ after stage (c). Furthermore, with probability at least $1-\varepsilon$, no two nonconsecutive b-cubes of A receive nontrivial input.

In what follows, we complete the proof by showing that A is sorted after stage (h) whenever (i) A is b-sorted after stage (a), and (ii) every b-cube of A is sorted after stage (c). (Note that these conditions are satisfied with probability at least $1-\varepsilon-2\cdot\varepsilon'$.)

Accordingly, assume that conditions (i) and (ii) hold. Then A is sorted after stage (c) with the exception of at most two nontrivial b-cubes of A. If A contains 0 or 1 nontrivial b-cubes after stage (c), then A is easily seen to be sorted after stages (c), (d), and (h). If there are 2 nontrivial b-cubes in A after stage (c) then they are adjacent. If b-cubes $2\cdot j$ and $2\cdot j+1$ are nontrivial for some integer j, $0 \le j < \mathrm{pow}(a-b-1)$, then A is sorted after stages (d) and (h). If b-cubes $2\cdot j+1$ and $2\cdot j+2$ are nontrivial for some integer j, $0 \le j < \mathrm{pow}(a-b-1)-1$, then stage (d) has no effect and the output of stage (h) is sorted. □

Lemmas 4.3, 4.6, and 6.2 together imply that

$$\mathit{Sort}_D(d,a,\varepsilon) \le \mathit{Sort}_D(d,a,b,\varepsilon) + O(b) \tag{10}$$

for all a, b, and d such that $0 \le b \le a \le d$, and all ε in $[0,1]$.

LEMMA 6.3. *For all a, b, and d such that $0 \le b \le a \le d$, and all ε in $[0,1]$, we have*

$$\mathit{Sort}_D(d, a, b+1, \varepsilon) \le \mathit{Most}_D(d, a, b, \varepsilon) + \mathit{Insert}_D(d, a-b).$$

Proof. We argue that a (d,a)-network in $\mathit{Sort}_N(d, a, b+1, \varepsilon)$ can be constructed by composing the following: (a) any (d,a)-network in $\mathit{Most}_N(d, a, b, \varepsilon)$; (b) an appropriate d-permutation π in $\Pi(d, a)$; (c) the d-permutation $\hookrightarrow_d^b$; (d) any $(d, a-b)$-network in $\mathit{Insert}_N(d, a-b)$; and (e) the d-permutation $\hookleftarrow_d^b$. (Note that this construction does indeed give a (d,a)-network.)

We may assume that the input is an a-random 0-1 d-vector. By the definition of $\mathit{Most}_N(d, a, b, \varepsilon)$, there is some d-permutation π in $\Pi(d, a)$ such that each a-cube is b-mostly-sorted with respect to π with probability at least $1-\varepsilon$ after stage (a). This is the desired stage (b) d-permutation π. Consider an arbitrary a-cube A. In what follows, we complete the proof by showing that if A is b-mostly-sorted with respect to π after stage (a), then A is $(b+1)$-sorted after stage (e).

Accordingly, let us assume that A is b-mostly-sorted with respect to π after stage (a). Then each $(a-b)$-cube of A contains an insertion instance after stage (c). Thus, each $(a-b)$-cube of A is sorted after stage (d). Furthermore, note that each $(a-b)$-cube of A contains the same number of 0's to within 2. Hence A has a dirty region of size at most $2 \cdot \mathrm{pow}(b) = \mathrm{pow}(b+1)$ after stage (e), as desired. □

LEMMA 6.4. *For all a, b, and d such that $0 \le b \le a \le d$, and all γ, ε, and ε' in $[0,1]$, we have*

$$\begin{aligned}\mathit{Sort}_D(d, a, b'+3, \varepsilon + 2\cdot\varepsilon') \le \mathit{Sort}_D(d, a, b, \varepsilon) + \mathit{Sort}_D(d, b+2, b'+1, \varepsilon') \\ + \mathit{Merge}_D(d, b-b', 1),\end{aligned}$$

where $b' = \lfloor \gamma \cdot (b+2) \rfloor$.

Proof. We may assume that $b > b'+3$ and $a > b+2$, since the claim is trivial otherwise. Let $m = \mathrm{pow}(a-b-2)$. We argue that a (d,a)-network in $\mathit{Sort}_N(d, a, b'+3, \varepsilon + 2\cdot\varepsilon')$ can be constructed by composing the following: (a) any (d,a)-network in $\mathit{Sort}_N(d, a, b, \varepsilon)$; (b) a random d-permutation drawn from $\Pi_R(d, b+2)$; (c) any $(d, b+2)$-network in $\mathit{Sort}_N(d, b+2, b'+1, \varepsilon')$; (d) an appropriate d-permutation π in $\Pi(d, a)$; (e) any $(d, b-b'+1)$-network in $\mathit{Merge}_N(d, b-b', 1)$; and (f) an appropriate d-permutation π' in $\Pi(d, a)$. (Note that this construction does indeed give a (d,a)-network.)

We may assume that the input is an a-random 0-1 d-vector. By the definition of $\mathit{Sort}_N(d, a, b, \varepsilon)$, each a-cube is b-sorted with probability at least $1-\varepsilon$ after stage (a). The output of stage (b) is $(b+2)$-random by Lemma 4.9. Hence, each $(b+2)$-cube is $(b'+1)$-sorted with probability at least $1-\varepsilon'$ after stage (c). Consider an arbitrary a-cube A, and let B_i denote the ith $(b+2)$-cube of A, $0 \le i < m$.

After stage (c), note that the following claims hold with probability at least $1-\varepsilon-2\cdot\varepsilon'$: (i) the dirty region of A has size at most $\mathrm{pow}(b)+2\cdot\mathrm{pow}(b'+1) \le \mathrm{pow}(b+1)$, and (ii) every B_i is $(b'+1)$-sorted. Let B_i^- (resp., B_i^+) denote $(b+1)$-cube 0 (resp., 1) of B_i. (In other words, B_i^- and B_i^+ are the "low half" and "high half" of B_i, respectively.) If condition (i) holds, then the dirty region of A is either confined to some B_i, or to some (B_{i-1}^+, B_i^-) pair, $0 < i < m$. Let us say that case 1 holds if, after stage (c), the dirty region of A is confined to some B_i and conditions (i) and (ii) hold. Similarly, case 2 holds if, after stage (c), the dirty region of A is confined to some (B_{i-1}^+, B_i^-) pair and conditions (i) and (ii) hold. Otherwise, case 3 holds. Note that case 3 holds with probability at most $\varepsilon + 2\varepsilon'$.

We now define the d-permutation π to be applied in stage (d). Break each $(b+1)$-cube B_i^+ (resp., B_i^-) into $\mathrm{pow}(b'+1)$ equal-sized sets $B_{i,j}^+$ (resp., $B_{i,j}^-$) by applying the $(b+1)$-permutation $\hookrightarrow_{b+1}^{b'+1}$ and then partitioning into $(b-b')$-cubes. (In other words, the set $B_{i,j}^+$ consists of those wires in B_i^+ with indices congruent to j modulo $\mathrm{pow}(b'+1)$, $0 \le j < \mathrm{pow}(b'+1)$.) In the arguments that follow, let C_i denote $B_{i-1}^+ \cup B_i^-$ and C_i^j denote the jth $(b-b'+1)$-cube of C_i, $0 < i < m$, $0 \le j < \mathrm{pow}(b'+1)$. The d-permutation π is defined in such a way that: (i) the wires in B_0^- and B_{m-1}^+ are left alone, and (ii) for each (B_{i-1}^+, B_i^-) pair, $0 < i < m$, the wires of $B_{i-1,j}^+$ and $B_{i,j}^-$ are brought into opposite halves of $(b-b'+1)$-cube C_i^j in preparation for the merge step of stage (e), $0 \le j < \mathrm{pow}(b'+1)$.

If either case 1 or case 2 holds after stage (c), note that the following claims hold after stage (d), $0 < i < m$; (i) all of the $B_{i-1,j}^+$'s and $B_{i,j}^-$'s are sorted, and (ii) all of the $B_{i-1,j}^+$'s (resp., $B_{i,j}^-$'s) have the same number of 0's to within 2.

If either case 1 or case 2 holds after stage (c), note that the following claims hold after stage (e), $0 < i < m$: (i) every C_i^j is sorted, and (ii) all of the C_i^j's have the same number of 0's to within 4.

We now define the d-permutation π' of stage (f) so that: (i) the wires in B_0^- and B_{m-1}^+ are left alone, and (ii) for each i, $0 \le i < m-1$, the C_i^j's are interleaved (in place) by applying the $(b+1)$-permutation $\hookleftarrow_{b+1}^{b'+1}$.

Let $C_0 = B_0^-$, $C_m = B_{m-1}^+$, and assume that either case 1 or case 2 held after stage (c). Then the following conditions hold after stage (f): (i) C_i has a dirty region of size at most $4 \cdot \mathrm{pow}(b'+1) = \mathrm{pow}(b'+3)$, $0 \le i \le m$, and (ii) no two nonconsecutive C_i's are nontrivial. If 0 or 1 of the C_i's are nontrivial then A is $(b'+3)$-sorted, and we are done. Otherwise, we can assume that C_i and C_{i+1} are nontrivial for some particular i, $0 \le i < m$. It follows easily that case 2 held after stage (c), and that the output of A after stage (f) is the same as after stage (c); hence, the dirty region of A has size at most $\mathrm{pow}(b'+1)$ after stage (f). □

Lemmas 4.4, 5.17, and 6.3 together imply that

$$Sort_D(d, a, \lfloor \gamma \cdot a \rfloor + 1, O(\mathrm{pow}(a) \cdot \varepsilon)) \le 2 \cdot a - \lfloor \gamma \cdot a \rfloor$$

for any admissible triple (γ, ε, a) and all d such that $0 \le a \le d$. Lemma 5.6 implies that for any function $\varepsilon(a) = \mathrm{pow}(-\mathrm{pow}(o(a)))$, there is a function $f(a) = o(1)$ such that $(\gamma_c + f(a), \varepsilon, a)$ is an admissible triple for all $a \ge 0$. Hence,

$$Sort_D(d, a, \lfloor \gamma(a) \cdot a \rfloor + 1, \mathrm{pow}(-\mathrm{pow}(o(a)))) \le 2 \cdot a - \lfloor \gamma(a) \cdot a \rfloor, \tag{11}$$

with $\gamma(a) = \gamma_c + o(1)$. Substituting the bounds of equation (11) (with $a = b+2$) and Lemma 4.3 (with $(a, d) = (b - \lfloor \gamma \cdot (b+2) \rfloor + 1, d)$) into the inequality of Lemma 6.4, we find that

$$\begin{aligned} &Sort_D(d, a, \lfloor (\gamma_c + o(1)) \cdot (b+2) \rfloor + 3, \mathrm{pow}(-\mathrm{pow}(o(b))) + \varepsilon') \\ &\le Sort_D(d, a, b, \varepsilon') + (3 - 2 \cdot \gamma_c + o(1)) \cdot b, \end{aligned}$$

for all ε' in $[0, 1]$. Starting with equation (11), and then iteratively applying the preceding inequality (with $b \approx \gamma \cdot a, \gamma^2 \cdot a, \gamma^3 \cdot a, \ldots$), until equation (10) can be "inexpensively" applied (e.g., with $b = o(a)$) we find that

$$Sort_D(d, a, \mathrm{pow}(-\mathrm{pow}(o(a)))) \le \frac{2 - \gamma_c^2 + o(1)}{1 - \gamma_c} \cdot a$$

for all a and d such that $0 \leq a \leq d$. Using Lemma 4.8 to eliminate the random aspects of the preceding construction, the proof of Lemma 6.1 is now complete. (We remark that randomization has not been used in the operation phase of any level in our construction. Furthermore, the only nontrivial probability distributions used in the permutation phase of any level are the $\Pi_R(d, a)$ distributions, $0 < a \leq d$.)

Note that we have used the AKS sorting network as part of our construction. It should be emphasized, however, that the AKS sorting network is only used to allow the function $\varepsilon(d)$ of Lemma 6.1 to be set as small as possible. For example, one could prove Lemma 6.1 with $\varepsilon(d) = \mathrm{pow}(-\mathrm{pow}(o(\sqrt{d})))$ by cutting off the preceding recurrence at $b = o(\sqrt{d})$ and applying bitonic sort, instead of cutting it off at $b = o(d)$ and applying the AKS sorting network.

7. An optimal-depth hypercubic network that sorts most inputs. In this section, we establish the existence of a depth-$O(d)$, hypercubic d-network that sorts most inputs. Once again we make use of the high-level strategy described at the beginning of section 6, except that we make use of Lemma 5.17 instead of Lemma 5.16. In contrast with section 6, however, we do not concern ourselves with constant factor issues. This leads to a much simpler construction. In particular, we do not require a hypercubic analogue of Lemma 6.4.

LEMMA 7.1. *Let $\psi(a)$ be any function such that $Sort^h_D(d, \psi(a), 0) = O(d)$, $0 \leq a \leq d$. For each function $\varepsilon(d) = \mathrm{pow}(-\mathrm{pow}(\psi(d)))$, we have*

$$Sort^h_D(d, a, \varepsilon(a)) = O(a).$$

Furthermore, there is a deterministic d-network that achieves this bound. □

By Lemma 4.5, the function ψ appearing in the statement of Lemma 7.1 is $\Omega(\sqrt{d})$. In fact, by Theorem 9.1, we have

$$\psi(d) = \Omega\left(\frac{d}{\mathrm{pow}(\sqrt{\upsilon_c \cdot \lg d}) \cdot \lg d}\right). \tag{12}$$

The following theorem provides an interpretation of Lemma 7.1 in the permutation domain.

THEOREM 7.1. *Let the function ψ be as defined in Lemma 7.1. For each function $\varepsilon(d) = \mathrm{pow}(-\mathrm{pow}(\psi(d)))$, there is a deterministic hypercubic d-network of depth $O(d)$ that sorts a random d-permutation drawn from $\Pi_R(d)$ with probability at least $1 - \varepsilon(d)$.*

Proof. The proof is similar to that of Theorem 6.1. □

LEMMA 7.2. *For all a, b, and d such that $0 \leq b \leq a \leq d$, and all ε and ε' in $[0, 1]$, we have*

$$Sort^h_D(d, a, \varepsilon + 2 \cdot \varepsilon') \leq Sort^h_D(d, a, b, \varepsilon) + Sort^h_D(d, b, \varepsilon') + 2 \cdot Merge^h_D(d, b, 1) + O(a).$$

Proof. The proof is similar to that of Lemma 6.2. The only difference is that we use an $O(a)$-depth hypercubic (d, a)-network (guaranteed to exist by Lemma 4.7) to implement each of the d-permutations of stages (b), (e), (g), and (h). This accounts for the additive $O(a)$ term on the right-hand side of the inequality. □

LEMMA 7.3. *For all a, b, and d such that $0 \leq b \leq a \leq d$, and all ε in $[0, 1]$, we have*

$$Sort^h_D(d, a, b + 1, \varepsilon) \leq Most^h_D(d, a, b, \varepsilon) + Insert^h_D(d, a - b) + O(a).$$

Proof. The proof is similar to that of Lemma 6.3. The only difference is that we use an $O(a)$-depth hypercubic (d, a)-network (guaranteed to exist by Lemma 4.7) to implement each of the d-permutations of stages (b), (c), and (e). This accounts for the additive $O(a)$ term on the right-hand side of the inequality. □

Lemmas 4.4, 5.17, and 7.3 together imply that

$$Sort_D^h(d, a, \lfloor \gamma \cdot a \rfloor + 1, O(\mathrm{pow}(a) \cdot \varepsilon)) = O(a) \tag{13}$$

for all d and any admissible triple (γ, ε, a) such that $0 \leq a \leq d$. Let δ be any constant, $0 < \delta < 1 - \gamma_c$. By Lemma 5.6, there is a function $g(d) = \Theta(d)$ such that $(\gamma_c + \delta, \mathrm{pow}(-\mathrm{pow}(g(a))), a)$ is an admissible triple for all $a \geq 0$. Hence,

$$Sort_D^h(d, a, \lfloor \gamma \cdot a \rfloor + 1, \mathrm{pow}(-\mathrm{pow}(\Theta(a)))) = O(a),$$

with $\gamma = \gamma_c + \delta$, and $0 \leq a \leq d$. Lemmas 4.3 and 7.2 (with $b = \lfloor \gamma \cdot a \rfloor + 1$) now give

$$Sort_D^h(d, a, \mathrm{pow}(-\mathrm{pow}(\Theta(a))) + 2 \cdot \varepsilon') \leq Sort_D^h(d, \lfloor \gamma \cdot a \rfloor + 1, \varepsilon') + O(a).$$

for all ε' in $[0, 1]$. Iteratively applying the preceding inequality, we find that

$$Sort_D^h(d, a, \mathrm{pow}(-\mathrm{pow}(\Theta(b)))) \leq Sort_D^h(d, b, 0) + O(a)$$

for all a, b, and d such that $0 \leq b \leq a \leq d$. Substituting $\psi(a)$ for b, where the function ψ is as defined in the statement of Lemma 7.1, we obtain

$$Sort_D^h(d, a, \mathrm{pow}(-\mathrm{pow}(\psi(a)))) = O(a)$$

for all a and d such that $0 \leq a \leq d$. Using Lemma 4.8 to eliminate the random aspects of the preceding construction, the proof of Lemma 7.1 is now complete. (We remark that randomization has not been used in the operation phase of any level in our construction. Furthermore, the only use of randomization in the permutation phase arises from applying Lemma 4.7 to implement random d-permutations drawn from $\Pi_R(d, a)$, $0 < a \leq d$.)

8. Deterministic merging. Many sorting algorithms, both sequential as well as parallel, are based on merging. For instance, sequential merge sort and Batcher's bitonic sorting network are both based on 2-way merging. Since merging two sorted lists of length $\mathrm{pow}(d)$ requires $\Omega(d)$ depth, one cannot hope to obtain a $o(d^2)$-depth sorting network (hypercubic or otherwise) by repeated 2-way merging. This section describes how to use a network $\mathcal{N}$ that sorts most inputs to construct a high-order merging network $\mathcal{N}'$, that is, an m-way merging network for some $m \gg 2$. A similar technique has recently been used by Ajtai, Komlós, and Szemerédi as part of an improved version of their original sorting network construction [3]. The multiplicative constant associated with the new construction is significantly lower than the constant established by Paterson [15], though it remains impractical.

The main idea underlying the results of this section may be informally outlined as follows. An n-input network is an m-way merging network if and only if it correctly merges all possible input vectors consisting of m sorted vectors of length n/m. But as in the case of sorting networks, we can easily prove the following 0-1 principle for merging networks: Any n-input network that correctly merges every input vector consisting of m sorted 0-1 vectors of length n/m is an m-way merging network. A key observation is that the total number of 0-1 vectors which can be obtained by

concatenating m sorted 0-1 vectors of length n/m is $(n/m+1)^m$, which is fairly small (compared to 2^n, the total number of 0-1 vectors, for example) when m is not too large. As a result, we can use an averaging argument to prove that, given any network $\mathcal{N}$ which for all k sorts a sufficiently high fraction (dependent on m) of the $\binom{n}{k}$ 0-1 vectors with k 0's and $n-k$ 1's, there exists a permutation π such that the network $\mathcal{N}'$ obtained by composing π with $\mathcal{N}$ sorts *all* of the $(n/m+1)^m$ 0-1 vectors consisting of m sorted 0-1 vectors of length n/m; in other words, $\mathcal{N}'$ is an m-way merging network with the same size and depth as network $\mathcal{N}$.

LEMMA 8.1. *Let $\mathcal{N}$ denote a (hypercubic) (d,a)-network that sorts each a-cube with probability at least $1-\varepsilon$ on any a-random* 0-1 *input d-vector, $m=\mathrm{pow}(d-a)$, Φ denote a subset of $\Phi(d)$, $\Phi^{(i)}\subseteq\Phi(a)$ denote the projection of the ith a-cube of Φ onto $\Phi(a)$, $0\le i<m$, and $\Phi'=\cup_{0\le i<m}\Phi^{(i)}$. If $|\Phi'|<1/\varepsilon$, then there exists a d-permutation π in $\Pi(d,a)$ such that the (hypercubic) (d,a)-network $\mathcal{N}'$ obtained by composing π with $\mathcal{N}$ satisfies*

$$\Phi\subseteq \mathit{Sort}(\mathcal{N}',a).$$

Proof. By definition, (hypercubic) (d,a)-network $\mathcal{N}$ can be partitioned along input a-cubes into m disjoint, identical a-networks $\mathcal{N}_a$. Thus, it is sufficient to construct an a-permutation π' such that the (hypercubic) a-network $\mathcal{N}'_a$ obtained by composing π' with $\mathcal{N}_a$ satisfies

$$\Phi'\subseteq \mathit{Sort}(\mathcal{N}'_a).$$

To determine a suitable a-permutation π', we construct the undirected bipartite graph with vertex sets $U=\Phi'$ and $V=\Pi(a)$, and an edge from vertex ϕ in U to vertex π in V if and only if the 0-1 a-vector ϕ is sorted by the a-network obtained by composing a-permutation π with $\mathcal{N}_a$. The degree of every vertex in U is at least $(1-\varepsilon)\cdot(\mathrm{pow}(a))!$, and so the sum of the degrees of the vertices in U is at least $(1-\varepsilon)\cdot|U|\cdot(\mathrm{pow}(a))!$. This sum is identical to that attained over V, so some vertex π' in V has degree at least $(1-\varepsilon)\cdot|U|>|U|-1$. Since the degree of π' is an integer, vertex π' is connected to every vertex in U. Thus, the a-network obtained by composing a-permutation π' with network $\mathcal{N}_a$ sorts every element of Φ'. □

LEMMA 8.2. *For nonnegative integers a' and b', let $\mathcal{N}$ be defined as in Lemma* 8.1 *with $a=a'+b'$ and*

$$(\mathrm{pow}(a')+1)^{\mathrm{pow}(b')}<1/\varepsilon.$$

Let Φ denote the set of all 0-1 *d-vectors ϕ such that each a-cube of ϕ belongs to $\Phi_M(a',b')$. Then there exists a d-permutation π in $\Pi(d,a)$ such that the (hypercubic) (d,a)-network $\mathcal{N}'$ obtained by composing π with $\mathcal{N}$ satisfies*

$$\Phi\subseteq \mathit{Sort}(\mathcal{N}',a).$$

Proof. We can apply Lemma 8.1 (with $\Phi'=\Phi_M(a',b')$) since

$$\begin{aligned}|\Phi_M(a',b')| &= (\mathrm{pow}(a')+1)^{\mathrm{pow}(b')}\\ &< 1/\varepsilon.\end{aligned}$$ □

Note that in Lemmas 8.1 and 8.2, the depth of $\mathcal{N}'$ exceeds that of $\mathcal{N}$ only by the depth required to implement a d-permutation in $\Pi(d,a)$. By Lemma 4.7, this additional depth is at most $2\cdot a$ for a hypercubic construction. (For a nonhypercubic construction, no additional depth is required to implement a fixed permutation.)

9. A near-optimal hypercubic sorting network. In this section, we construct a hypercubic sorting network with nearly logarithmic depth. At a high level, the construction is simply based on recursive high-order merging: the input is partitioned into some number of equal-sized lists, each of these lists is sorted recursively, and the resulting set of sorted lists are merged together. The recursion is cut off by applying bitonic sort on subproblems that are sufficiently small. The primary question that remains to be addressed is how to perform the merge step efficiently. Lemmas 9.1 and 9.2 make use of the results of section 8 (specifically, Lemma 8.2) to reduce the merge step to a smaller sorting problem.

LEMMA 9.1. *Let $\nu(d)$ be any function such that $\nu(d) = \omega(1)$ and $\nu(d) = o(d)$, and let $\varepsilon(d) = \mathrm{pow}(-\mathrm{pow}(\mu(d) \cdot d))$ where $\mu(d) = \mu_c - \frac{1}{\nu(d)}$. For all a, b, and d such that $0 \le b \le a \le d$, we have*

$$Sort_D^h(d, a, O(\mathrm{pow}(b) \cdot \varepsilon(b))) \le Sort_D^h(d, b, 0) + O(a \cdot \nu(a)).$$

Proof. By Lemma 5.7, there exists a function $\gamma(d) = 1 - \frac{3-o(1)}{4 \cdot \nu(d)}$ such that $(\gamma(a), \varepsilon(a), a)$ is an admissible triple for all $a \ge 0$. For such an admissible triple, equation (13) then implies

$$Sort_D^h(d, a, \lfloor \gamma(a) \cdot a \rfloor + 1, O(\mathrm{pow}(a) \cdot \varepsilon(a))) = O(a).$$

Lemmas 4.3 and 7.2 now give

$$Sort_D^h(d, a, O(\mathrm{pow}(a) \cdot \varepsilon(a)) + 2 \cdot \varepsilon') \le Sort_D^h(d, \lfloor \gamma(a) \cdot a \rfloor + 1, \varepsilon') + O(a),$$

for all ε' in $[0, 1]$. Iteratively applying the preceding inequality, we find that

$$Sort_D^h(d, a, O(\mathrm{pow}(b) \cdot \varepsilon(b))) \le Sort_D^h(d, b, 0) + O(a \cdot \nu(a))$$

for all a, b, and d such that $0 \le b \le a \le d$. □

LEMMA 9.2. *Let the function ν be as defined in Lemma 9.1, and let $\mu(d) = \mu_c - \frac{2}{\nu(d)}$. Then*

$$Merge_D^h(d, a - \lfloor \mu(b) \cdot b \rfloor, \lfloor \mu(b) \cdot b \rfloor) \le Sort_D^h(d, b, 0) + O(a \cdot \nu(a))$$

for all a, b, and d such that $(2 + \lg a) \cdot \nu(b) \le b \le a \le d$.

Proof. Let $\varepsilon(d) = \mathrm{pow}(-\mathrm{pow}(\mu'(d) \cdot d))$ where $\mu'(d) = \mu_c - \frac{1}{\nu(d)}$. By Lemma 9.1,

$$Sort_D^h(d, a, O(\mathrm{pow}(b) \cdot \varepsilon(b))) \le Sort_D^h(d, b, 0) + O(a \cdot \nu(a))$$

for all a, b, and d such that $0 \le b \le a \le d$. Hence, for b sufficiently large, there exists a hypercubic (d, a)-network $\mathcal{N}$ of depth $Sort_D^h(d, b, 0) + O(a \cdot \nu(a))$ that sorts each a-cube with probability at least $1 - \varepsilon'$ on any a-random 0-1 input d-vector, where

$$\begin{aligned} \varepsilon' &= \mathrm{pow}(2 \cdot b) \cdot \varepsilon(b) \\ &= \mathrm{pow}(\mathrm{pow}(1 + \lg b) - \mathrm{pow}(\mu'(b) \cdot b)) \\ &< \mathrm{pow}(-\mathrm{pow}(\mu'(b) \cdot b - 1)). \end{aligned}$$

The result now follows from Lemma 8.2 since

$$\begin{aligned} (\mathrm{pow}(a - \lfloor \mu(b) \cdot b \rfloor) + 1)^{\mathrm{pow}(\lfloor \mu(b) \cdot b \rfloor)} &< \mathrm{pow}(2 \cdot a \cdot \mathrm{pow}(\mu(b) \cdot b)) \\ &= \mathrm{pow}(\mathrm{pow}(\mu(b) \cdot b + 1 + \lg a)) \\ &= \mathrm{pow}(\mathrm{pow}(\mu'(b) \cdot b + 1 + \lg a - b/\nu(b))) \\ &\le \mathrm{pow}(\mathrm{pow}(\mu'(b) \cdot b - 1)) \\ &< 1/\varepsilon'. \end{aligned}$$

□

A recurrence for $Sort^h_D(d, a, 0)$ can be developed using the preceding lemma. Note that

$$Sort^h_D(d, a, 0) \leq \min_{0 \leq b \leq a} Sort^h_D(d, a-b, 0) + Merge^h_D(d, a-b, b)$$

for all a, b, and d such that $0 \leq b \leq a \leq d$. (This inequality is immediate, since we can always sort a-cubes by: (i) sorting $(a-b)$-cubes, and (ii) merging the sorted $(a-b)$-cubes within each a-cube.) Let the functions ν and μ be as defined in Lemma 9.2. Applying Lemma 9.2 to the preceding inequality, we obtain

$$Sort^h_D(d, a, 0) \leq \min_{(2+\lg a)\cdot\nu(b) \leq b \leq a} Sort^h_D(d, a - \lfloor \mu(b) \cdot b \rfloor, 0) + Sort^h_D(d, b, 0) + O(a \cdot \nu(a))$$

for all a, b, and d such that $0 \leq b \leq a \leq d$. Fixing d and letting $S(a) = Sort^h_D(d, a, 0)$, we can write this recurrence more simply as

$$S(a) \leq \min_{(2+\lg a)\cdot\nu(b) \leq b \leq a} S(a - \lfloor \mu(b) \cdot b \rfloor) + S(b) + O(a \cdot \nu(a)).$$

In Appendix A it is proven that

$$S(a) = O(a \cdot \mathrm{pow}(\sqrt{\upsilon_c \cdot \lg a}) \cdot \lg a).$$

(The constant υ_c is defined in equation (9).) The preceding bound is proven with $\nu(d) = \Theta(\sqrt{\lg d})$, and seems to be the best upper bound obtainable using this recurrence. Setting $a = d$, we obtain a proof of the following theorem.

THEOREM 9.1. *For all $d \geq 0$, we have*

$$Sort^h_D(d, 0) = O(d \cdot \mathrm{pow}(\sqrt{\upsilon_c \cdot \lg d}) \cdot \lg d). \qquad \square$$

10. An optimal randomized hypercubic sorting algorithm. In section 7, we constructed a depth-$O(d)$ hypercubic sorting network that sorts most d-permutations. In the present section, we modify that result to obtain a polynomial-time uniform $O(d)$-depth, coin-tossing hypercubic network that sorts *every* d-permutation (and hence, every d-vector) with high probability. We then use this coin-tossing network to develop a polynomial-time uniform hypercubic algorithm that sorts every d-vector in $O(d)$ time with high probability.

We define a *hypercubic algorithm* as any normal hypercube algorithm. (See [12, section 3.1.3], for example, for a definition of the class of normal hypercube algorithms.) Every depth-a hypercubic sorting d-network corresponds to a (possibly nonuniform) hypercubic sorting algorithm that runs in $O(a)$ time on any $\mathrm{pow}(d)$-processor hypercubic machine. Of course, the converse is not true in general; most of the basic operations allowed within a normal hypercube algorithm (e.g., the usual set of arithmetic operations) cannot be performed by a hypercubic sorting network.

A sorting network is hard-wired and has a fixed depth, or "running time," that is independent of the input. On the other hand, a sorting algorithm can have an arbitrarily large gap between its worst-case and average-case running times. For example, consider a sorting algorithm with the following structure:

1. Apply a random d-permutation drawn from $\Pi_R(d)$ to the input d-vector.
2. Attempt to sort the resulting d-vector using a time-$T(d)$ method that correctly sorts most d-permutations.

3. Check whether step 2 was successful. If so, halt. If not, return to step 1.

The worst-case running time of such an algorithm is infinite, while the average-case running time could be as low as $O(T(d))$. One might attempt to develop a hypercubic sorting algorithm with this structure by using the d-network of Theorem 7.1 to implement step 2 with $T(d) = O(d)$. Step 3 is trivial to implement in $O(d)$ time. However, two difficulties remain to be addressed.

The first difficulty is that step 1 is not easily implemented by a hypercubic algorithm. We will overcome this difficulty by making use of a depth-d shuffle-"?" d-network to randomly permute the input data. Although the d-permutation π applied by such a d-network is not d-random, we prove in Lemma 10.2 below that π is sufficiently random for our purposes.

The second difficulty is that the hypercubic algorithm corresponding to Theorem 7.1 is not polynomial-time uniform. This undesirable characteristic stems from our use of Lemma 4.8 to remove the random aspects of our hypercubic network construction. As discussed at the end of section 7, there is only one source of randomness in our construction: Whenever we apply the shuffle-"+" (d, a)-network of Lemma 5.17, we first apply an a-random d-permutation. We overcome the second difficulty by replacing such a-random d-permutations with a depth-a unshuffle-"0" d-network followed by a depth-a shuffle-"?" d-network.

Lemmas 10.2 and 10.3 below prove that we can approximately sort *every* d-permutation with high probability by applying a depth-d shuffle-"?" d-network followed by a depth-$\lceil d/2 \rceil$ shuffle-"+" d-network. Lemma 10.1 widens the range of input distributions for which the analysis of section 5 can be applied. (Recall that the σ_α functions were defined in section 5.1.)

LEMMA 10.1. *Let $\mathcal{N}$ denote a coin-tossing d-network, and assume that for each length-d binary string α, input wire α is set to* 0 *with probability p_α, and to* 1 *otherwise. Further assume that the set of events $E_\alpha \stackrel{\text{def}}{=}$ "input wire α receives a* 0" *are mutually independent. Then the output wire with index α receives a* 0 *with probability at least $\sigma_\alpha(p^-)$ and at most $\sigma_\alpha(p^+)$, where $p^- = \min_\alpha p_\alpha$ and $p^+ = \max_\alpha p_\alpha$.*

Proof. The proof is immediate from Lemma 4.11. □

LEMMA 10.2. *Let a, b, and d denote nonnegative integers such that $d = a + b$, (γ, ε, a) be an admissible triple,*

$$\delta = (1 - 2 \cdot \varepsilon) \cdot \mathrm{pow}((\gamma - 1) \cdot a)/4,$$
$$\varepsilon' = 2 \cdot e^{-2 \cdot \delta^2 \cdot \mathrm{pow}(b)},$$

and $\mathcal{N}$ denote the depth-$(a + d)$ d-network obtained by composing the following: (i) *a depth-d shuffle-"?" d-network, and* (ii) *a depth-a shuffle-"+" d-network. Then there exists a set A of at least* $\mathrm{pow}(d) - \mathrm{pow}(\gamma \cdot a + b)$ *output wires of $\mathcal{N}$, and a fixed permutation π of A, such that the following condition holds with probability at least* $1 - \mathrm{pow}(d) \cdot \varepsilon - \mathrm{pow}(a) \cdot \varepsilon'$ *after execution of $\mathcal{N}$ on any* 0-1 *input d-vector ϕ: If permutation π is applied to A, then the resulting length-$|A|$* 0-1 *output vector is $(\gamma \cdot a + b)$-sorted.*

Proof. Let the 0-1 vector ϕ in $\Phi(d, k)$ be input to d-network $\mathcal{N}$, and set $p = k/\mathrm{pow}(d)$. Throughout this proof, the symbols α and β will be used to denote binary strings of length a and b, respectively. A *random execution* will refer to an execution of d-network $\mathcal{N}$ on input ϕ.

For each β, define $C_0(\beta)$ (resp., $C_2(\beta)$, $C_3(\beta)$) as the set of $\mathrm{pow}(a)$ level-0 input wires (resp., level-d input wires, level-$(a + d - 1)$ output wires) with indices of the

form $\alpha\beta$ (resp., $\alpha\beta$, $\beta\alpha$) for some α. For each α, define $C_1(\alpha)$ as the set of $\text{pow}(b)$ level-$(a-1)$ output wires with indices of the form $\beta\alpha$ for some β.

For each β, define p_β to be the fraction of 0's induced by input ϕ on $C_0(\beta)$ (i.e., the number of 0's assigned to $C_0(\beta)$ divided by $\text{pow}(a)$). Note that $p = (\sum_\beta p_\beta)/\text{pow}(b)$. For each α, let X_α denote the random variable corresponding to the number of 0's received by the wires of $C_1(\alpha)$ in a random execution, and let $q_\alpha = X_\alpha/\text{pow}(b)$. Note that, unlike the p_β's, each q_α is a random variable. Furthermore, the random variable X_α is easily seen to be the sum of $\text{pow}(b)$ independent Bernoulli trials, where trial β has success probability p_β. Thus, a standard Chernoff-type argument [6] implies

$$\Pr\{|X_\alpha - p \cdot \text{pow}(b)| \geq \vartheta \cdot \text{pow}(b)\} \leq 2 \cdot e^{-2 \cdot \vartheta^2 \cdot \text{pow}(b)} \tag{14}$$

for all $\vartheta \geq 0$. Define a random execution to be *δ-balanced* if

$$p - \delta \leq q_\alpha \leq p + \delta$$

for all α. By equation (14), a random execution is δ-balanced with probability at least $1 - \text{pow}(a) \cdot \varepsilon'$ (set $\vartheta = \delta$.)

Note that the last a levels of d-network $\mathcal{N}$ form a (d, a)-network. Hence, these levels can be partitioned into $\text{pow}(b)$ disjoint depth-a shuffle-"+" a-networks $\mathcal{N}_\beta$, where the input and output wires of $\mathcal{N}_\beta$ correspond to $C_2(\beta)$ and $C_3(\beta)$, respectively. Let $E_{\alpha\beta}$ denote the event that input α of $\mathcal{N}_\beta$ (i.e., level-d input wire $\alpha\beta$ of $\mathcal{N}$) receives a 0 in a random execution. Let $f_{\alpha\beta}(p)$ denote the probability that $E_{\alpha\beta}$ occurs in a random δ-balanced execution. Let $g_{\beta\alpha}(p)$ denote the probability that output α of $\mathcal{N}_\beta$ (i.e., output $\beta\alpha$ of $\mathcal{N}$) receives a 0 in a random δ-balanced execution. Note that $f_{\alpha\beta}(p) = q_\alpha$, since wire α of $C_2(\beta)$ receives the value of a wire chosen uniformly at random from $C_1(\alpha)$. Furthermore, since the sets $C_1(\alpha)$ are mutually disjoint, we find that for each β and for each fixed setting of the q_α values, the $\text{pow}(a)$ events $E_{\alpha\beta}$ are mutually independent. Lemma 10.1 can therefore be applied to each a-network $\mathcal{N}_\beta$ and yields

$$\sigma_\alpha(p - \delta) \leq g_{\beta\alpha}(p) \leq \sigma_\alpha(p + \delta)$$

for all α and β.

Define A to be the set (guaranteed to exist by Lemma 5.13) of at least $\text{pow}(d) - \text{pow}(\gamma \cdot a + b)$ output wires of $\mathcal{N}$ indexed by length-d binary strings $\beta\alpha$ such that

$$\Gamma_\alpha(1 - \varepsilon) - \Gamma_\alpha(\varepsilon) \leq \delta.$$

We set π to the permutation of set A that sorts the $\Gamma_\alpha(\varepsilon)$ values in ascending order. Ties may be broken arbitrarily. (As discussed in the proof of Lemma 5.14, the set A and permutation π can be computed efficiently.) It remains to prove that our choice of A and π satisfies the requirements of the lemma.

Let $p^- = \max\{0, p - 2 \cdot \delta\}$, $p^+ = \min\{1, p + 2 \cdot \delta\}$, and B denote the set of binary strings α in A for which $\Gamma_\alpha(\varepsilon)$ is contained in $[p^-, p + \delta]$. Because the σ_α's are monotonically increasing, and using linearity of expectation, we have

$$\begin{aligned}
\sum_\alpha |\sigma_\alpha(p^+) - \sigma_\alpha(p^-)| &= \sum_\alpha \big(\sigma_\alpha(p^+) - \sigma_\alpha(p^-)\big) \\
&= \left(\sum_\alpha \sigma_\alpha(p^+)\right) - \left(\sum_\alpha \sigma_\alpha(p^-)\right) \\
&= (p^+ - p^-) \cdot \text{pow}(a) \\
&\leq 4 \cdot \delta \cdot \text{pow}(a).
\end{aligned}$$

For each α in B we have $\sigma_\alpha(p^-) \leq \varepsilon$, $\sigma_\alpha(p^+) \geq 1-\varepsilon$, and hence

$$|\sigma_\alpha(p^+) - \sigma_\alpha(p^-)| \geq 1 - 2 \cdot \varepsilon.$$

The preceding inequalities imply that

$$\begin{aligned} |B| &\leq 4 \cdot \delta \cdot \mathrm{pow}(a+b)/(1-2\cdot\varepsilon) \\ &= \mathrm{pow}(\gamma \cdot a + b). \end{aligned}$$

Note that the set of binary strings B is mapped to a contiguous interval by permutation π. Let A^- (resp., A^+) denote the set of all binary strings in $A \setminus B$ mapped to positions lower (resp., higher) than B by π.

Let $\beta\alpha$ denote the binary string associated with an arbitrary output in A^-. Thus $\Gamma_\alpha(\varepsilon) < p^-$ (so $p^- > 0$ and $p - \delta > 0$), which implies $\Gamma_\alpha(1-\varepsilon) < p - \delta$ and hence $\sigma_\alpha(p-\delta) > 1-\varepsilon$. Combining this inequality with the lower bound of equation (10), we find that $g_{\beta\alpha}(p) > 1-\varepsilon$. (The probability that output $\beta\alpha$ receives a 1 is less than ε.)

Similarly, let α denote the binary string associated with some output in A^+. Thus $\Gamma_\alpha(\varepsilon) > p + \delta$ (so $p + \delta < 1$), which implies $\sigma_\alpha(p+\delta) < \varepsilon$. Combining this inequality with the upper bound of equation (10), we find that $g_{\beta\alpha}(p) < \varepsilon$. (The probability that output $\beta\alpha$ receives a 0 is less than ε.)

We conclude that if permutation π is applied to A, the resulting length-$|A|$ 0-1 vector has a dirty region of size at most $|B| = \mathrm{pow}(\gamma \cdot a + b)$ with probability at least $1 - \mathrm{pow}(d) \cdot \varepsilon - \mathrm{pow}(a) \cdot \varepsilon'$. □

LEMMA 10.3. *Let $\mathcal{N}$, A, and π be defined as in Lemma* 10.2*, with $a = \lceil d/2 \rceil$ and $b = \lfloor d/2 \rfloor$. Then there exist constants γ and ε in $(0,1)$ such that the following condition holds with probability at least $1 - O(\mathrm{pow}(-\mathrm{pow}(\varepsilon \cdot d)))$ after execution of $\mathcal{N}$ on any* 0-1 *input vector ϕ: If permutation π is applied to A, then the resulting length-$|A|$* 0-1 *output vector is $(\gamma \cdot d)$-sorted.*

Proof. This is a straightforward consequence of Lemmas 5.6 and 10.2. □

Given Lemma 10.3, we can easily prove an analogue of Lemma 7.3 that uses "?" gates instead of random permutations. (It is important to note that the (d,a)-network associated with the construction of Lemma 10.3 is composed of the following four stages: (a) a depth-a unshuffle-"0" d-network; (b) a depth-a shuffle-"?" d-network; (c) a depth-$\lceil a/2 \rceil$ unshuffle-"0" d-network; and (d) a depth-$\lceil a/2 \rceil$ shuffle-"+" d-network.) We can then use the scheme of section 7 to prove Theorem 10.1 below with

$$\varepsilon(d) = \mathrm{pow}(-\mathrm{pow}(\Theta(\sqrt{d}))).$$

Unfortunately, we cannot take advantage of the improvement associated with equation (12) because the construction of section 9 is not polynomial-time uniform.

THEOREM 10.1. *For each function $\varepsilon(d) = \mathrm{pow}(-\mathrm{pow}(O(\sqrt{d})))$, there is a polynomial-time uniform $O(d)$-depth coin-tossing hypercubic d-network that sorts any input d-vector probability at least $1 - \varepsilon(d)$.*

The scheme of section 7 can also be used to prove Theorem 10.1 below with the function ε as defined in Theorem 10.2. In this case, we can dramatically decrease the failure probability by making use of the Sharesort algorithm of Cypher and Plaxton [9]. Sharesort is a polynomial-time uniform hypercubic sorting algorithm with worst-case running time $O(d \cdot \lg^2 d)$ [9]. Note that Sharesort runs in $O(d)$ time on $O(d/\lg^2 d)$-cubes. Hence, we can modify the scheme of section 7 by cutting off the sorting recurrence at $\Theta(d/\lg^2 d)$-cubes instead of $\Theta(\sqrt{d})$-cubes (as allowed by bitonic sort).

Unfortunately, Sharesort does not correspond to a hypercubic sorting network since, for example: (i) Sharesort makes copies of keys, and (ii) Sharesort performs a variety of arithmetic operations on auxiliary integer variables. For these reasons, we have not been able to make use of Sharesort in previous sections of the paper.

A small improvement to the Sharesort bound is known when polynomial-time "preprocessing" (to compute certain look-up tables) is allowed. In particular, the running time of Sharesort can be improved to $O(d \cdot (\lg d) \cdot \lg^* d)$ in that case [8]. This improvement has been incorporated into the $\varepsilon(d)$ bound of Theorem 10.2. If exponential preprocessing is allowed, the running time of Sharesort can be improved further to $O(d \cdot \lg d)$ [8]. However, it is not clear whether the latter result could be used to improve the $\varepsilon(d)$ bound of Theorem 10.2. (The lack of uniformity can be eliminated through randomization. However, the failure probability of the resulting algorithm seems to be strictly higher than that given by Theorem 10.2.)

THEOREM 10.2. *Let* $f(d) = \Theta(\frac{d}{(\lg d) \cdot \lg^* d})$. *For each function*

$$\varepsilon(d) = \text{pow}(-\text{pow}(f(d))),$$

there is a polynomial-time uniform randomized hypercubic sorting algorithm that runs in $O(d)$ *time* (*on any input* d*-vector*) *with probability at least* $1 - \varepsilon$.

11. An optimal bit-serial randomized hypercubic sorting algorithm. An *order-d omega machine* is a $((d+1) \cdot \text{pow}(d))$-processor machine, $d \geq 0$. Each processor has an associated ID of the form (i, j), $0 \leq i \leq d$, $0 \leq j < \text{pow}(d)$. We define the *ith level* of a given order-d omega machine as the set of $\text{pow}(d)$ processors with IDs of the form (i, j), $0 \leq j < \text{pow}(d)$. The processors of an order-d omega machine are interconnected according to the following rules:

1. There is no wire between any pair of processors in nonconsecutive levels.
2. For all i such that $0 \leq i < d$, there is a wire connecting processor (i, j) to processor $(i+1, j')$ if and only if $j_k = j'_{k+1}$, $0 \leq k < d-1$.

Omega machines belong to the class of butterfly-like machines discussed in [12, section 3.8.1].

Observe that there is a close correspondence between an order-d omega machine $\mathcal{M}$ and a depth-d shuffle d-network $\mathcal{N}$. In particular, consider the 1-1 function $f(i, j)$ that maps processor (i, j) of $\mathcal{M}$ to the following: (i) level-i input wire j of $\mathcal{N}$, $0 \leq i < d$, and (ii) level-$(i-1)$ output wire j of $\mathcal{N}$, $0 < i \leq d$. (Recall that level-i input wire j and level-$(i-1)$ output wire j represent the same wire, $0 < i < d$. Hence, f is indeed a function.) Then there is a wire between processors (i, j) and $(i+1, j')$ in $\mathcal{M}$ if and only if wires $f(i, j)$ and $f(i+1, j')$ in $\mathcal{N}$ are connected to a common gate x, with $f(i, j)$ as an input wire and $f(i+1, j')$ as an output wire.

In the bit model, it is assumed that a processor can only perform one bit operation per time step. Thus, b bit steps are required to send a b-bit message to an adjacent processor. Similarly, b bit steps are required to compare two b-bit operands located at the same processor. In this section, we provide a bit-serial polynomial-time uniform randomized algorithm for sorting $\text{pow}(d)$ $O(b)$-bit records on an order-d omega machine in $O(b + d)$ bit steps. This time bound is easily seen to be optimal. For $b = \Omega(d)$, the processor bound is also optimal. Our algorithm can be adapted to achieve the same asymptotic performance on any butterfly-like machine.

DEFINITION 11.1. *A* bit-serial omega emulation scheme *for a depth-a word-size-b d-network* $\mathcal{N}$ *is a bit-serial algorithm that* (i) *runs on an order-d omega machine,* (ii) *emulates the execution of* $\mathcal{N}$ *on any d-vector of b-bit integers, and* (iii) *receives (resp.,*

produces) the jth component of the input (resp., output) d-vector at processor (d, j), $0 \leq j < \mathrm{pow}(d)$.

We remark that the somewhat unnatural input-output convention used in the preceding definition (level d is used instead of level 0) is not essential. We could easily modify the results of this section to hold if the input and output are provided at level 0.

DEFINITION 11.2. *A depth-$(2 \cdot b)$ hypercubic d-network $\mathcal{N}$ is* a-pass, $0 \leq b \leq a \leq d$, *if and only if the d-permutation $\hookrightarrow_d$ (resp., $\hookleftarrow_d$) is applied in the permutation phase of the first (resp., last) b levels of $\mathcal{N}$.*

LEMMA 11.1. *There is an $O(a + b)$-bit-step bit-serial omega emulation scheme for any coin-tossing a-pass word-size-b d-network.*

Proof. The proof is straightforward. (Note that each of our five gate types can be implemented in a bit-serial fashion. For the "+" and "−" gates, such a bit-serial implementation requires that the inputs be provided most-significant-bit first.) □

DEFINITION 11.3. *A depth-$O(a)$ hypercubic d-network $\mathcal{N}$ is* a-multipass, $0 \leq a \leq d$, *if and only if $\mathcal{N}$ can be decomposed into an $O(1)$-length sequence of a-pass networks.*

LEMMA 11.2. *There is an $O(a + b)$-bit-step bit-serial omega emulation scheme for any coin-tossing a-multipass word-size-b d-network.*

Proof. This follows from a constant number of applications of Lemma 11.1. Note that only levels d through $d-a$ of the order-d omega machine are used by the emulation scheme. □

DEFINITION 11.4. *A hypercubic d-network $\mathcal{N}$ is* (γ, a, k)-geometric, $\gamma \geq 0$, $0 \leq a \leq d$, $k \geq 0$, *if and only if $\mathcal{N}$ can be decomposed into a length-k sequence of d-networks $\langle \mathcal{N}_i \rangle$ such that:* (i) $0 \leq \gamma \leq 1$ *and $\mathcal{N}_i$ is $\lfloor \gamma^{i+1} \cdot a \rfloor$-multipass, $0 \leq i < k$, or* (ii) $\gamma > 1$ *and $\mathcal{N}_i$ is $\lfloor \gamma^{i-k} \cdot a \rfloor$-multipass, $0 \leq i < k$.*

DEFINITION 11.5. *A (γ, a, k)-geometric d-network is* compact *if and only if*

$$\sum_{0 \leq i < k} \left(\lfloor \gamma^{i+1} \cdot a \rfloor + 1 \right) \leq d.$$

LEMMA 11.3. *There is an $O(b + d)$-bit-step bit-serial omega emulation scheme for any coin-tossing compact (γ, a, k)-geometric word-size-b d-network $\mathcal{N}$.*

Proof. We give the proof for the case $0 \leq \gamma \leq 1$. The case $\gamma > 1$ is similar. (Note that if we "reverse" a (γ, a, k)-geometric d-network, we obtain a $(1/\gamma, a, k)$-geometric d-network.)

Decompose the given d-network $\mathcal{N}$ into a sequence of networks $\langle \mathcal{N}_i \rangle$ as in Definition 11.4. Thus, network $\mathcal{N}_i$ is $(f(a, i) - 1)$-multipass where

$$f(a, i) = \lfloor \gamma^{i+1} \cdot a \rfloor + 1,$$

$0 \leq i < k$. Let $\mathcal{M}$ denote an order-d omega machine, and

$$g(i) = d - \sum_{0 \leq j < i} f(a, j),$$

$0 \leq i < k$. To obtain an efficient bit-serial omega emulation scheme for network $\mathcal{N}$, we use Lemma 11.2 to emulate $\mathcal{N}_i$ on the contiguous set of levels $A_i = \{g(i), g(i) - 1, \ldots, g(i+1)+1\}$ of $\mathcal{M}$, $0 \leq i < k$. (The input to $\mathcal{N}_i$ is provided at level $g(i)$. Hence, Lemma 11.2 implies that the output to network $\mathcal{N}_i$ is produced at level $g(i)$.) Note that (i) the A_i's are well defined (i.e., each is a set of level numbers in the range 0 to

d) since $\mathcal{N}$ is compact, and (ii) the A_i's are pairwise disjoint. Property (ii) ensures that the emulation schemes associated with distinct $\mathcal{N}_i$'s do not interfere with one another.

It remains to prove that we can efficiently communicate the output of network $\mathcal{N}_i$, which is produced at level $g(i)$, to level $g(i+1)$, $0 \leq i < k-1$. (Once the output of $\mathcal{N}_i$ reaches level $g(i+1)$, the emulation of $\mathcal{N}_{i+1}$ over the set of levels A_{i+1} can begin.) For example, it would suffice to efficiently map the value on wire j of level $g(i)$ to wire j of level $g(i+1)$, $0 \leq j < \text{pow}(d)$. Unfortunately, it is very expensive to implement such an identity d-permutation on the machine $\mathcal{M}$.

The key observation, however, is that we do not need to implement the identity d-permutation between levels $g(i)$ and $g(i+1)$. Because the $f(a,i)$'s form a non-increasing sequence, $0 \leq i < k$, there is no interaction between distinct $f(a,i+1)$-cubes in any d-network $\mathcal{N}_j$ such that $i+1 \leq j < k$. Hence, it is sufficient to implement a d-permutation π such that: (i) level-$g(i)$ $f(a,i+1)$-cubes are mapped to level-$g(i+1)$ $f(a,i+1)$-cubes, and (ii) the identity $f(a,i+1)$-permutation is applied within each $f(a,i+1)$-cube.

We now prove that a d-permutation π of the desired form can be implemented in $O(f(a,i)+f(a,i+1))$ bit steps while using only levels $A_i \cup A_{i+1}$ of $\mathcal{M}$. (The reader may wonder why the preceding O-bound does not depend on b. This bound refers only to the "additional" number of bit steps required to implement the d-permutation π. Because our entire emulation scheme is pipelined bit-serially, we can account for the word size by adding $O(b)$ at the end of our analysis.) In particular, the d-permutation π that we apply from level $g(i)$ to level $g(i+1)$ satisfies

$$\pi(j) = j_{f(a,i)+f(a,i+1)-1} \cdots j_{f(a,i+1)} j_{d-1} \cdots j_{f(a,i)+f(a,i+1)} j_{f(a,i+1)-1} \cdots j_0,$$

$0 \leq j < \text{pow}(d)$. The d-permutation π can easily be implemented in $O(f(a,i) + f(a,i+1))$ additional bit steps as follows. First, within level $g(i)$, we apply the d-permutation π' defined by

$$\pi'(j) = j_{d-1} \cdots j_{f(a,i)+f(a,i+1)} j_{f(a,i+1)-1} \cdots j_0 j_{f(a,i)+f(a,i+1)-1} \cdots j_{f(a,i+1)},$$

$0 \leq j < \text{pow}(d)$. Note that π' belongs to $\Pi(d, f(a,i)+f(a,i+1))$. By Lemma 4.7, π' can be implemented in $O(f(a,i)+f(a,i+1))$ bit steps using only levels $A_i \cup A_{i+1}$ of $\mathcal{M}$. We now complete the implementation of d-permutation π by repeatedly unshuffling the data from level $g(i)$ to level $g(i+1)$. (This takes $f(a,i)$ bit steps and uses only the set of levels A_i.)

The total running time of our emulation scheme is easily seen to be

$$O\left(b + \sum_{0 \leq i < k} f(a,i)\right) = O(b + d - g(k-1))$$

bit steps. However, one final detail remains to be addressed, since our emulation scheme leaves the output at level $g(k-1)$ instead of level d. We can move the output to level d by applying an appropriate fixed permutation π' from level $g(k-1)$ to level d. The entire machine $\mathcal{M}$ can be used for this purpose. Thus, the Beneš routing technique of Lemma 4.7 yields an immediate $O(b+d)$ bound. In fact, an $O(b+d-g(k-1))$ bound can easily be achieved by (i) shuffling the data from level $g(k-1)$ to level d, and (ii) applying an appropriate permutation π'' in $\Pi(d, d-g(k-1))$.

Thus, the total running time of our revised emulation scheme is $O(b+d-g(k-1))$, which is $O(b+d)$ since $\mathcal{N}$ is compact. □

THEOREM 11.1. *There is a polynomial-time uniform $O(b+d)$-bit-step bit-serial omega emulation of the family $\{\mathcal{N}_d\}$ of coin-tossing d-networks of Theorem* 10.1.

Proof. Examining the construction of Theorem 10.1, we find that $\mathcal{N}_d$ can be decomposed into the following: (a) a $(1, d, 1)$-geometric coin-tossing d-network; (b) a $(1/3, d, O(\lg d))$-geometric coin-tossing d-network; (c) a $(1, O(\sqrt{\lg d}), O(\sqrt{\lg d}))$-geometric deterministic d-network; (d) a $(3, d, O(\lg d))$-geometric coin-tossing d-network; and (e) a $(1, d, 1)$-geometric coin-tossing d-network. (Note that the constants $1/3$ and 3 appearing above are somewhat arbitrarily chosen. The "slack" in our definitions creates a range of acceptable values.)

The coin-tossing d-networks of stages (a), (b), (d), and (e) are easily seen to be compact. Stage (c) corresponds to the use of bitonic sort to cut off the sorting recurrence at $O(\sqrt{\lg d})$-cubes. Note that we are free to choose the constant within the $O(\sqrt{\lg d})$ bound arbitrarily small. Hence, we can ensure that the deterministic d-network of stage (c) is also compact. The theorem then follows by Lemma 11.3. (The emulation is polynomial-time uniform since (i) the family of coin-tossing networks of Theorem 10.1 is polynomial-time uniform, and (ii) the emulation scheme underlying Lemma 11.3 is polynomial-time uniform.) □

THEOREM 11.2. *For each function $\varepsilon(d) = \mathrm{pow}(-\mathrm{pow}(O(\sqrt{d})))$, there is a polynomial-time uniform $O(b+d)$-bit-step randomized bit-serial algorithm for an order-d omega machine that sorts any input d-vector of b-bit integers with probability at least $1 - \varepsilon(d)$.*

Proof. The claim is an immediate consequence of Theorems 10.1 and 11.1. □

12. Concluding remarks. We have defined the class of "hypercubic" sorting networks and established a nearly logarithmic upper bound on the depth complexity of such networks. Of course, it would be very interesting to close the remaining gap. Given the techniques developed in this paper, the problem of constructing an optimal $O(\lg n)$-depth hypercubic sorting network has been reduced to the problem of constructing an $O(\lg n)$-depth comparator network that sorts a randomly chosen input permutation with probability at least $1 - 2^{-n^{\varepsilon}}$ for some constant $\varepsilon > 0$.

One unfortunate characteristic of our hypercubic sorting network construction is its lack of uniformity. In particular, no deterministic polynomial-time algorithm is known for generating the family of networks for which existence has been established. On the positive side, existence of a randomized polynomial-time generation algorithm is a straightforward consequence of our results.

While the multiplicative constant of approximately 7.44 established for the sorting network construction of section 6 appears to be quite reasonable, the construction remains impractical. This is due to the fact that there is a trade-off between the value of the multiplicative constant and the success probability (the probability that a random input permutation is sorted by the network); for practical values of n, a significant increase in the constant is required in order to prove any reasonable success probability.

Appendix A. Analysis of the sorting recurrence. Let c_1 be a positive constant, let a_0 and c_2 denote sufficiently large positive constants (to be determined), and consider the function $T(a)$, $a \geq 0$, defined by the following recurrence. Let $T(a) = O(1)$ for $a \leq a_0$, and

$$T(a) = \min_{(2+\lg a)\cdot\nu(b) \leq b \leq a} T(a - \lfloor \mu(b) \cdot b \rfloor) + T(b) + c_1 \cdot a \cdot \nu(a) \tag{A.15}$$

for $a > a_0$, where

$$\nu(d) = c_2 \cdot \sqrt{\lg d},$$
$$\mu(d) = \mu_c - \frac{2}{\nu(d)}.$$

Note that our choice of the function $\nu(d)$ satisfies the requirements of Lemma 9.2, since $\nu(d) = \omega(1)$ and $\nu(d) = o(d)$. Furthermore, the function $\mu(d)$ is defined as in Lemma 9.2. Hence, any upper bound for $T(a)$ is also an upper bound for the function $S(a)$ defined towards the end of section 9. In this section, we prove that

$$T(a) = O(a \cdot \mathrm{pow}(\sqrt{\upsilon_c \cdot \lg a}) \cdot \lg a).$$

(The constant υ_c is defined in equation (9).) We prove two technical lemmas before analyzing the recurrence of equation (A.15).

LEMMA A.1. *For every pair of real numbers x and y such that $0 \le x \le y$, we have*

$$\sqrt{x \cdot y} - \sqrt{x \cdot y - x \cdot \sqrt{x \cdot y}} \ge x/2.$$

Proof. Examining the Taylor series expansion of $\sqrt{1-\varepsilon}$, we find that

$$\sqrt{1-\varepsilon} \le 1 - \varepsilon/2$$

for all ε in $[0, 1]$. Hence,

$$\begin{aligned}\sqrt{x \cdot y} - \sqrt{x \cdot y - x \cdot \sqrt{x \cdot y}} &= \sqrt{x \cdot y} - \sqrt{x \cdot y} \cdot \sqrt{1 - \sqrt{x/y}} \\ &\le \sqrt{x \cdot y} - \sqrt{x \cdot y} \cdot \left(1 - \frac{\sqrt{x}}{2 \cdot \sqrt{y}}\right) \\ &= x/2. \qquad \square\end{aligned}$$

LEMMA A.2. *For any $a > a_0$, let*

$$a' = a - \lfloor \mu(b') \cdot b' \rfloor,$$

where

$$b' = \left\lfloor \frac{a}{\mathrm{pow}(\sqrt{\upsilon_c \cdot \lg a})} \right\rfloor.$$

For a sufficiently large choice of the constant a_0, the following bounds hold:

(i) $1 < a' \le a - \frac{\mu(b') \cdot a}{\mathrm{pow}(\sqrt{\upsilon_c \cdot \lg a})} + 2 < a$,
(ii) $\mu(b') \ge \mu_c - \frac{4}{c_2 \cdot \sqrt{\lg a}}$,
(iii) $2 \cdot \mathrm{pow}(\sqrt{\upsilon_c \cdot \lg a}) = o(a/\sqrt{\lg a})$,
(iv) $1 < (2 + \lg a) \cdot \nu(b') \le b' < a$, *and*
(v) $\sqrt{\upsilon_c \cdot \lg a} - \sqrt{\upsilon_c \cdot \lg a - \upsilon_c \cdot \sqrt{\upsilon_c \cdot \lg a}} > \upsilon_c/2$.

Proof. Part (v) follows from Lemma A.1. The remaining inequalities follow from straightforward asymptotic estimates. $\square$

THEOREM A.1. *There is a positive constant c_0 such that*

$$T(a) \leq c_0 \cdot a \cdot \mathrm{pow}(\sqrt{\upsilon_c \cdot \lg a}) \cdot \lg a$$

for all $a > 1$.

Proof. We prove the claim of the lemma by induction on a. The claim is trivial for $1 < a \leq a_0$, since we are free to choose the constant c_0 arbitrarily large. Now fix $a > a_0$ and assume the claim holds for all smaller values of a. Define a' and b' as in Lemma A.2.

By part (i) of Lemma A.2, we can apply the induction hypothesis to bound $T(a')$ since $1 < a' < a$. Using parts (i), (ii), and (iii) of Lemma A.2, we find that

$$\begin{aligned} T(a') &\leq c_0 \cdot a' \cdot \mathrm{pow}(\sqrt{\upsilon_c \cdot \lg a'}) \cdot \lg a' \\ &\leq c_0 \cdot \left(a - \frac{\mu(b') \cdot a}{\mathrm{pow}(\sqrt{\upsilon_c \cdot \lg a})} + 2\right) \cdot \mathrm{pow}(\sqrt{\upsilon_c \cdot \lg a}) \cdot \lg a \\ &\leq c_0 \cdot a \cdot \mathrm{pow}(\sqrt{\upsilon_c \cdot \lg a}) \cdot \lg a - c_0 \cdot \mu_c \cdot a \cdot \lg a + c_0 \cdot \left(\frac{4}{c_2} + o(1)\right) \cdot a \cdot \sqrt{\lg a}. \end{aligned}$$

By part (iv) of Lemma A.2, we can apply the induction hypothesis to bound $T(b')$ since $1 < b' < a$. Using part (v) of Lemma A.2 and equation (9), we find that

$$\begin{aligned} T(b') &\leq c_0 \cdot b' \cdot \mathrm{pow}(\sqrt{\upsilon_c \lg b'}) \cdot \lg b' \\ &\leq c_0 \cdot a \cdot \mathrm{pow}\left(\sqrt{\upsilon_c \cdot \lg a - \upsilon_c \cdot \sqrt{\upsilon_c \cdot \lg a}} - \sqrt{\upsilon_c \cdot \lg a}\right) \cdot \left(\lg a - \sqrt{\upsilon_c \cdot \lg a}\right) \\ &\leq c_0 \cdot \mu_c \cdot a \cdot \lg a - c_0 \cdot \mu_c \cdot a \cdot \sqrt{\upsilon_c \cdot \lg a}. \end{aligned}$$

By part (iv) of Lemma A.2, we can set $b = b'$ in the recurrence of equation (A.15) since $(2 + \lg a) \cdot \nu(b') \leq b' < a$. Thus,

$$\begin{aligned} T(a) &\leq T(a') + T(b') + c_1 \cdot a \cdot \nu(a) \\ &\leq c_0 \cdot a \cdot \mathrm{pow}(\sqrt{\upsilon_c \cdot \lg a}) \cdot \lg a \\ &\quad + \left[c_0 \cdot \left(\frac{4}{c_2} + o(1) - \mu_c \cdot \sqrt{\upsilon_c}\right) + c_1 \cdot c_2\right] \cdot a \cdot \sqrt{\lg a}. \end{aligned}$$

For a_0 sufficiently large, we can choose the constants c_0 and c_2 so that the coefficient of $a\sqrt{\lg a}$ is negative, yielding

$$T(a) \leq c_0 \cdot a \cdot \mathrm{pow}(\sqrt{\upsilon_c \cdot \lg a}) \cdot \lg a,$$

as required. ∎

Appendix B. Locating the "median" of the binomial distribution. The purpose of this appendix is to establish Theorem B.1, which is used in the proof of Lemma 5.15.

LEMMA B.1. *Let k and n be integers such that $0 \leq k \leq n$, p be a real number in $[0,1]$ such that $np = k$, and X be a random variable drawn from $B(n,p)$ (i.e., the binomial distribution with parameters n and p). Let Y be a random variable corresponding to the number of successes in n independent Bernoulli trials with associated success probabilities p_i, $0 \leq i < n$, such that $\sum_{0 \leq i < n} p_i = k$. Then*

$$\begin{aligned} \Pr(X < k) &\geq \Pr(Y < k), \\ \Pr(X > k) &\geq \Pr(Y > k). \end{aligned}$$

Proof. These inequalities are established (in a more general form) by Hoeffding in [10, Theorem 4]. □

LEMMA B.2. *Let k, n, p, and X be as defined in Lemma* B.1. *If $0 \le p \le 1/2$, then let Y' and Y'' be random variables drawn from $B(2k, 1/2)$ and $B(n-2k, 0)$, respectively. Otherwise, let Y' and Y'' be random variables drawn from $B(2(n-k), 1/2)$ and $B(2k-n, 1)$, respectively. Let $Y = Y' + Y''$. If*

$$\Pr(X = k) \ge \Pr(Y = k)/2,$$

then

$$\min\{\Pr(X \le k), \Pr(X \ge k)\} \ge 1/2.$$

Proof. By symmetry, $\Pr(Y \le k) = \Pr(Y \ge k) = [1 + \Pr(Y = k)]/2$. The claimed inequality then follows easily using Lemma B.1. □

LEMMA B.3. *For each pair of integers k and n such that $0 \le k \le n$, let*

$$u_{k,n} = \binom{n}{k} \left(\frac{k}{n}\right)^k \left(1 - \frac{k}{n}\right)^{n-k}.$$

(If $n = k$, then $u_{k,n} = 1$.) Then $u_{k,n} \ge \frac{k^k}{e^k k!}$.

Proof. Fix $k \ge 0$, and let $v_n = u_{k,n}$ for $n \ge k$. It is sufficient to prove that the sequence $\langle v_n \rangle$ is nonincreasing for $n \ge k$ and that

$$\lim_{n \to \infty} v_n = \frac{k^k}{e^k k!}. \tag{B.16}$$

To see that the sequence $\langle v_n \rangle$ is nonincreasing for $n \ge k$, note that

$$\begin{aligned} \frac{v_{k+1}}{v_k} &= \left(\frac{k}{k+1}\right)^k \\ &\le 1, \end{aligned}$$

and for $n > k$ we have

$$\begin{aligned} \frac{v_{n+1}}{v_n} &= \left(1 + \frac{1}{n-k}\right)^{n-k} \Big/ \left(1 + \frac{1}{n}\right)^n \\ &\le 1. \end{aligned}$$

(The preceding inequality follows from the fact that the function $f : \mathbf{R} \to \mathbf{R}$ defined by $f(x) = (1 + \frac{1}{x})^x$ is strictly increasing for all $x \ge 1$.)

To verify equation (B.16), note that

$$\begin{aligned} v_n &= \frac{n!}{k!(n-k)!} \cdot \frac{k^k}{n^k} \cdot \frac{(n-k)^{n-k}}{n^{n-k}} \\ &= \frac{k^k}{k!} \cdot \frac{n!}{(n-k)!} \cdot \frac{(n-k)^{n-k}}{n^n} \\ &\sim \frac{k^k}{k!} \left(1 - \frac{k}{n}\right)^n \\ &\sim \frac{k^k}{e^k k!}. \end{aligned}$$

□

LEMMA B.4. *For each nonnegative integer k, let*

$$w_k = \frac{k^{k+1}2^{2k}k!}{(k+1)e^k(2k)!}.$$

Then the sequence $\langle w_k \rangle$ is nondecreasing for $k \geq 1$.

Proof. Note that

$$\frac{w_{k+1}}{w_k} = \frac{2(1+\frac{1}{k})^k(k+1)^3}{ek(k+2)(2k+1)}$$

and

$$\lim_{k\to\infty} \frac{w_{k+1}}{w_k} = 1.$$

Thus, it is sufficient to prove that the function $f : \mathbf{R} \to \mathbf{R}$ defined by

$$f(x) = \frac{(1+\frac{1}{x})^x(x+1)^3}{x(x+2)(2x+1)}$$

is nonincreasing for $x \geq 1$. One may easily verify that

$$\frac{df(x)}{dx} = \frac{(x+1)^2(1+\frac{1}{x})^x}{x^2(x+2)^2(2x+1)^2} \cdot g(x),$$

where

$$g(x) = -x^2 - 6x - 2 + x(x+1)(x+2)(2x+1)\left[\ln\left(1+\frac{1}{x}\right) - \frac{1}{x+1}\right].$$

Hence, for $x > 0$, $\frac{df(x)}{dx} \leq 0$ if and only if $g(x) \leq 0$. For $x \geq 1$, we have $\ln(1+\frac{1}{x}) \leq \frac{1}{x} - \frac{1}{2x^2} + \frac{1}{3x^3}$, and hence

$$\begin{aligned} g(x) &\leq -x^2 - 6x - 2 + x(x+1)(x+2)(2x+1)\left(\frac{1}{x(x+1)} - \frac{1}{2x^2} + \frac{1}{3x^3}\right) \\ &= -\frac{23x}{6} - \frac{7}{6} + \frac{4}{3x} + \frac{2}{3x^2} \\ &< 0. \qquad \square \end{aligned}$$

LEMMA B.5. *For all nonnegative integers k, we have*

$$\frac{k^k}{e^k k!} > \binom{2k}{k} 2^{-2k-1}.$$

Proof. It is easy to verify that the claim holds for $0 \leq k \leq 2$. For $k \geq 3$, consider the sequence $\langle w_k \rangle$ defined in Lemma B.4. By Lemma B.4, $w_k \geq w_3 \approx 0.538 > 1/2$ for $k \geq 3$. Hence

$$\begin{aligned} \frac{k^k}{e^k k!} &> \frac{k+1}{k}\binom{2k}{k}2^{-2k-1} \\ &> \binom{2k}{k}2^{-2k-1}. \qquad \square \end{aligned}$$

LEMMA B.6. *Let k, n, p, X, and Y be as defined in Lemma* B.2. *Then*

$$\Pr(X = k) > \Pr(Y = k)/2.$$

Proof. If $0 \le p \le 1/2$, then

$$\begin{aligned}\Pr(X = k) &= \binom{n}{k}\left(\frac{k}{n}\right)^k\left(1 - \frac{k}{n}\right)^{n-k}\\ &\ge \frac{k^k}{e^k k!}\\ &> \binom{2k}{k}2^{-2k-1}\\ &= \Pr(Y = k)/2,\end{aligned}$$

where the two inequalities follow from Lemmas B.3 and B.5, respectively.

Similarly, if $1/2 < p \le 1$, we have

$$\begin{aligned}\Pr(X = k) &= \binom{n}{k}\left(\frac{k}{n}\right)^k\left(1 - \frac{k}{n}\right)^{n-k}\\ &= \binom{n}{n-k}\left(\frac{n-k}{n}\right)^{n-k}\left(1 - \frac{n-k}{n}\right)^{n-(n-k)}\\ &\ge \frac{(n-k)^{n-k}}{e^{n-k}(n-k)!}\\ &> \binom{2(n-k)}{n-k}2^{-2(n-k)-1}\\ &= \Pr(Y = k)/2. \qquad \square\end{aligned}$$

LEMMA B.7. *Let k, n, p, and X be as defined in Lemma* B.1. *Then*

$$\min\{\Pr(X \le k), \Pr(X \ge k)\} \ge 1/2.$$

Proof. The proof is immediate from Lemmas B.2 and B.6. $\square$

THEOREM B.1. *Let n be a nonnegative integer, p be a real number in $[0, 1]$, and X be a random variable drawn from $B(n, p)$. Then*

$$\min\{\Pr(X \ge \lfloor np \rfloor), \Pr(X \le \lceil np \rceil)\} \ge 1/2.$$

Proof. Define real numbers p^- and p^+ so that $np^- = \lfloor np \rfloor$ and $np^+ = \lceil np \rceil$. Let X^- (resp., X^+) denote a random variable drawn from $B(n, p^-)$ (resp., $B(n, p^+)$). Note that for all real numbers x, we have

$$\begin{aligned}\Pr(X \ge x) &\ge \Pr(X^- \ge x),\\ \Pr(X \le x) &\ge \Pr(X^+ \le x).\end{aligned}$$

Combining these inequalities with the bound of Lemma B.7, we obtain

$$\begin{aligned}\Pr(X \ge \lfloor np \rfloor) &\ge \Pr(X^- \ge \lfloor np \rfloor) \ge 1/2,\\ \Pr(X \le \lceil np \rceil) &\ge \Pr(X^+ \le \lceil np \rceil) \ge 1/2. \qquad \square\end{aligned}$$

Acknowledgments. Thanks to Rostislav Caha, Don Coppersmith, Ming Kao, Sheralyn Listgarten, Yuan Ma, Bruce Maggs, and Torsten Suel for valuable discussions. We would also like to thank the anonymous referees who provided numerous valuable comments for improving the presentation of the paper.

REFERENCES

[1] W. Aiello, F. T. Leighton, B. Maggs, and M. Newman, *Fast algorithms for bit-serial routing on a hypercube*, Math. Systems Theory, 24 (1991), pp. 253–271.

[2] M. Ajtai, J. Komlós, and E. Szemerédi, *Sorting in $c \log n$ parallel steps*, Combinatorica, 3 (1983), pp. 1–19.

[3] M. Ajtai, J. Komlós, and E. Szemerédi, *Halvers and expanders*, in Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 686–692.

[4] K. E. Batcher, *Sorting networks and their applications*, in Proceedings of the AFIPS Spring Joint Computer Conference, 32 (1968), pp. 307–314.

[5] V. E. Beneš, *Optimal rearrangeable multistage connecting networks*, Bell System Technical Journal, 43 (1964), pp. 1641–1656.

[6] H. Chernoff, *A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations*, Ann. Math. Statist., 23 (1952), pp. 493–509.

[7] R. E. Cypher, *Theoretical aspects of VLSI pin limitations*, SIAM J. Comput., 22 (1993), pp. 58–63.

[8] R. E. Cypher and C. G. Plaxton, *Techniques for Shared Key Sorting*, Tech. Report RJ 7347, Computer Science Department, IBM Almaden Research Center, San Jose, CA, 1990.

[9] R. E. Cypher and C. G. Plaxton, *Deterministic sorting in nearly logarithmic time on the hypercube and related computers*, J. Comput. Systems Sci., 47 (1993), pp. 501–548.

[10] W. Hoeffding, *On the distribution of the number of successes in independent trials*, Ann. Math. Statist., 27 (1956), pp. 713–721.

[11] D. E. Knuth, *The Art of Computer Programming*, Vol. 3, Addison-Wesley, Reading, MA, 1973.

[12] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes*, Morgan-Kaufmann, San Mateo, CA, 1991.

[13] F. T. Leighton, B. M. Maggs, A. G. Ranade, and S. B. Rao, *Randomized routing and sorting on fixed-connection networks*, J. Algorithms, 17 (1994), pp. 157–205.

[14] F. T. Leighton and C. G. Plaxton, *A (fairly) simple circuit that (usually) sorts*, in Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 264–274.

[15] M. S. Paterson, *Improved sorting networks with $O(\log N)$ depth*, Algorithmica, 5 (1990), pp. 75–92.

[16] C. G. Plaxton, *A hypercubic sorting network with nearly logarithmic depth*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, ACM, New York, 1992, pp. 405–416.

[17] C. G. Plaxton and T. Suel, *A lower bound for sorting networks based on the shuffle permutation*, Math. Systems Theory, 27 (1994), pp. 491–508.

[18] A. G. Ranade, *How to emulate shared memory*, J. Comput. Systems Sci., 42 (1991), pp. 307–326.

[19] J. H. Reif and L. G. Valiant, *A logarithmic time sort for linear size networks*, J. Assoc. Comput. Mach., 34 (1987), pp. 60–76.

[20] L. G. Valiant and G. J. Brebner, *Universal schemes for parallel communication*, in Proceedings of the 13th Annual ACM Symposium on Theory of Computing, ACM, New York, 1981, pp. 263–277.

SIAM J. COMPUT.
Vol. 27, No. 1, pp. 48–64, February 1998
© 1998 Society for Industrial and Applied Mathematics
002

THE SHRINKAGE EXPONENT OF DE MORGAN FORMULAS IS 2*

JOHAN HÅSTAD†

Abstract. We prove that if we hit a de Morgan formula of size L with a random restriction from R_p, then the expected remaining size is at most $O(p^2(\log \frac{1}{p})^{3/2}L + p\sqrt{L})$. As a corollary we obtain an $\Omega(n^{3-o(1)})$-formula-size lower bound for an explicit function in P. This is the strongest known lower bound for any explicit function in NP.

Key words. lower bounds, formula size, random restrictions, computational complexity

AMS subject classification. 68Q25

PII. S0097539794261556

1. Introduction.

Proving lower bounds for various computational models is of fundamental value to our understanding of computation. Still, we are very far from proving strong lower bounds for realistic models of computation, but at least there is more or less constant progress. In this paper we study formula size for formulas over the basis $\wedge$, $\vee$, and $\neg$. Our technique is based on random restrictions which were first defined and explicitly used in [2], although some earlier results can be formalized in terms of this type of random restrictions.

To create a random restriction in the space R_p we, independently for each variable, keep it as a variable with probability p and otherwise assign it the value 0 or 1 with equal probabilities $\frac{1-p}{2}$. Now suppose we have a function given by a de Morgan formula of size L. What will be the expected formula size of the induced function when we apply a random restriction from R_p? The obvious answer is that this size will be at most pL.

Subbotovskaya [11] was the first to observe that actually formulas shrink more. Namely, she established an upper bound

$$O(p^{1.5}L + 1) \tag{1}$$

on the expected formula size of the induced function. This result allowed her to derive an $\Omega(n^{1.5})$ lower bound on the de Morgan formula size of the parity function.

This latter bound was superseded by Khrapchenko [12, 13], who, using a different method, proved a tight $\Omega(n^2)$ lower bound for the parity function. His result implied that the parity function shrinks by a factor $\theta(p^2)$, and provided an upper bound $\Gamma \leq 2$ on the *shrinkage exponent* Γ, defined as the least upper bound of all γ that can replace 1.5 in (1).

New impetus for research on the expected size of the reduced formula was given by Andreev [9] who, based upon Subbotovskaya's result, derived an $n^{2.5-o(1)}$ lower bound on the de Morgan formula size for a function in P. An inspection of the proof reveals that his method actually gives the bound $n^{\Gamma+1-o(1)}$ for the same function.

New improvements of the lower bound on Γ followed. Nisan and Impagliazzo [5] proved that $\Gamma \geq \frac{21-\sqrt{73}}{8} \approx 1.55$. Paterson and Zwick [6], complementing the technique from [5] with very clever and natural arguments, pushed this bound further to $\Gamma \geq \frac{5-\sqrt{3}}{2} \approx 1.63$.

* Received by the editors January 14, 1994; accepted for publication (in revised form) December 18, 1995.
http://www.siam.org/journals/sicomp/27-1/26155.html
† Royal Institute of Technology, Stockholm, Sweden (johanh@nada.kth.se).

One can also define the corresponding notion for read-once formulae. For such formulas it was established in [3] that $\Gamma_{ro} = 1/\log_2(\sqrt{5}-1) \approx 3.27$. This result was made tight in [1], in that a polylogarithmic factor in the bounds was removed.

In this paper we continue (and possibly end) this string of results by proving that $\Gamma = 2$. To be more precise we prove that the remaining size is $O(p^2(\log \frac{1}{p})^{3/2}L + p\sqrt{L})$. As discussed above, this gives an $\Omega(n^{3-o(1)})$ lower bound for the formula size of the function defined by Andreev.

Our proof is shown by a sequence of steps. We first analyze the probability of reducing the formula to a single literal. When viewing the situation suitably, this first lemma gives a nice and not very difficult generalization of Khrapchenko's [12, 13] lower bounds for formula size. As an illustration of the power of this lemma we next show, without too much difficulty, how to establish the desired shrinkage when the formula is balanced. The general case is more complicated due to the fact that we need to rely on more dramatic simplifications. Namely, suppose that $\phi = \phi_1 \wedge \phi_2$ and ϕ_1 is much smaller than ϕ_2. Then, from an intuitive point of view, it seems like we are in a good position to prove that we have substantial shrinkage since it seems quite likely that ϕ_1 is reduced to the constant 0 and we can erase all of ϕ_2. The key new point in the main proof is that we have to establish that this actually happens. In the balanced case, we did not need this mechanism. The main theorem is established in two steps. First we prove that the probability that a formula of size at least 2 remains after we have applied a restriction from R_p is small, and then we prove that the expected remaining size is indeed small.

It is curious to note that all except our last and main result are proved under arbitrary but "favorable" conditioning, while we are not able to carry this through for the main theorem.

2. Notation. A *de Morgan formula* is a binary tree in which each leaf is labeled by a literal from the set $\{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$ and each internal node v is labeled by an operation which is either $\wedge$ or $\vee$. The *size* of a formula ϕ is defined as the number of leaves and is denoted by $L(\phi)$. The *depth* $D(\phi)$ is the depth of the underlying tree. The *size* and the *depth* of a Boolean function f are, respectively, the minimal size and depth of any de Morgan formula computing f in the natural sense. For convenience we define the size and depth of a constant function as 0.

A *restriction* is an element of $\{0, 1, *\}^n$. For $p \in [0, 1]$ a random restriction ρ from R_p is chosen by randomly and independently setting each variable to $*$ with probability p and to $0, 1$ with equal probabilities $\frac{1-p}{2}$. The interpretation of giving the value $*$ to a variable is that it remains a variable, while in the other cases the given constant is substituted as the value of the variable.

All logarithms in this paper are to the base 2. We use the notation x_i^ϵ to denote x_i when $\epsilon = 0$ and $\bar{x}_i$ when $\epsilon = 1$. We also need the concept of a filter.

DEFINITION 2.1. *A set of restrictions Δ is a* filter *if, when $\rho \in \Delta$ and $\rho(x_i) = *$, then for $\epsilon \in \{0, 1\}$ the restriction ρ' obtained by setting $\rho'(x_j) = \rho(x_j)$, for every $j \neq i$ and $\rho'(x_i) = \epsilon$ also belongs to Δ.*

For any event E we will use $Pr[E|\Delta]$ as shorthand for $Pr[E|\rho \in \Delta]$. Note that the intersection of two filters is a filter.

3. Preliminaries. We analyze the expected size of a formula after it has been hit with a restriction from R_p. The variables that are given values are substituted into the formula after which we use the following rules of simplification:

- If one input to a $\vee$-gate is given the value 0, we erase this input and let the other input of this gate take the place of the output of the gate.
- If one input to a $\vee$-gate is given the value 1 we replace the gate by the constant 1.
- If one input to a $\wedge$-gate is given the value 1, we erase this input and let the other input of this gate take the place of the output of the gate.
- If one input to a $\wedge$-gate is given the value 0, we replace the gate by the constant 0.
- If one input of a $\vee$-gate is reduced to the single literal x_i ($\bar{x}_i$), then $x_i = 0$ (1) is substituted in the formula, giving the other input to this gate. If possible we do further simplifications in this subformula.
- If one input of a $\wedge$-gate is reduced to the single literal x_i ($\bar{x}_i$), then $x_i = 1$ (0) is substituted in the formula, giving the other input to this gate. If possible we do further simplifications in this subformula.

We call the last two rules the *one-variable simplification rules*. All rules preserve the function that the formula is computing. Observe that the one-variable simplification rules are needed to obtain a nontrivial decrease of the size of the formula, as can be seen from the pathological case in which the original formula consists of an $\vee$ (or $\wedge$) of L copies of a single variable x_i. If these rules did not exist, then with probability p the entire formula would remain and we could get an expected remaining size which is pL. Using the above rules we prove that instead we always get an expected remaining size which is at most slightly larger than p^2L.

In order to be able to speak about the reduced formula in an unambiguous way, let us be more precise about the order in which we do the simplification. Suppose that ϕ is a formula and that $\phi = \phi_1 \wedge \phi_2$. We first make the simplification in ϕ_1 and ϕ_2 and then only later make the simplifications which are connected with the top gate. This implies that the simplified ϕ will not always consist of a copy of simplified ϕ_1 and a copy of simplified ϕ_2, since the combination might give more simplifications. In particular, this will happen if ϕ_1 is simplified to one variable x_i since $x_i = 1$ will be substituted in the simplified ϕ_2. Whenever a one-variable simplification rule actually results in a change in the other subformula, we say that a one-variable simplification is *active* at the corresponding gate.

We let $\phi\lceil_\rho$ denote a formula that results when the above simplifications are done to ϕ. As usual, $L(\phi\lceil_\rho)$ denotes the size of this formula.

It is important to note that simplifications have a certain commutativity property. We say that two restrictions ρ_1 and ρ_2 are *compatible* if they never give two different constant values to the same x_i. In other words, for any x_i the pair $(\rho_1(x_i), \rho_2(x_i))$ is one of the pairs $(*,*), (*,0), (*,1), (0,*), (0,0), (1,*)$, or $(1,1)$. For compatible restrictions we can define the combined restriction $\rho_1 \circ \rho_2$ which, in the mentioned seven cases, takes the values $*, 0, 1, 0, 0, 1, 1$, respectively. This combined restriction is the result (on the variable level) of first doing ρ_1 and then doing ρ_2 on the variables given $*$ by ρ_1. Note that the fact that ρ_1 and ρ_2 are compatible makes the combining operator commutative. We need to make sure that combination also acts in the proper way on formulas.

LEMMA 3.1. *Let ρ_1 and ρ_2 be two compatible restrictions. Then for any ϕ,*

$$(\phi\lceil_{\rho_1})\lceil_{\rho_2} = (\phi\lceil_{\rho_2})\lceil_{\rho_1} = \phi\lceil_{\rho_1\circ\rho_2}.$$

Proof. Let ρ denote $\rho_1 \circ \rho_2$. Clearly, we need only establish $(\phi\lceil_{\rho_1})\lceil_{\rho_2} = \phi\lceil_\rho$ since the other equality follows by symmetry. We proceed by induction over the size of

ϕ. When the size is 1 the claim is obvious. For the general case we assume that $\phi = \phi_1 \wedge \phi_2$ (the case $\phi = \phi_1 \vee \phi_2$ being similar) and we have two cases:

1. Some one-variable simplification rule is active at the top gate when defining ϕ_{ρ_1}.
2. This is not the case.

In the second case, there is no problem since there is no interaction between ϕ_1 and ϕ_2 until after both ρ_1 and ρ_2 have been applied. Namely, by induction $\phi_j\lceil_\rho = (\phi_j\lceil_{\rho_1})\lceil_{\rho_2}$ for $j = 1, 2$, and since we do all simplification to the subformulas before we do any simplification associated with the top gate, the result will be the same in both cases.

In the first case, assume that $\phi_1\lceil_{\rho_1} = x_j^\epsilon$. This means that $x_j = \bar{\epsilon}$ is substituted in $\phi_2\lceil_{\rho_1}$. Viewing this as a restriction $\rho^{(j)}$ giving a non-$*$ value to only one variable, ϕ_2 is simplified to $(\phi_2\lceil_{\rho_1})\lceil_{\rho^{(j)}}$. Now we have three possibilities depending on the value of ρ_2 on x_j.

When $\rho_2(x_j) = \epsilon$, then $(\phi_1\lceil_{\rho_1})\lceil_{\rho_2} \equiv 0$ and by induction $\phi_1\lceil_\rho \equiv 0$ and hence $(\phi\lceil_{\rho_1})\lceil_{\rho_2} = \phi\lceil_\rho \equiv 0$.

When $\rho_2(x_j) = \bar{\epsilon}$, then by induction $(\phi_1\lceil_{\rho_1})\lceil_{\rho_2} = \phi_1\lceil_\rho \equiv 1$, and since $\rho^{(j)}$ is compatible with (even a subassignment of) ρ_2 we have

$$(\phi\lceil_{\rho_1})\lceil_{\rho_2} = ((\phi_2\lceil_{\rho_1})\lceil_{\rho^{(j)}})\lceil_{\rho_2} = (\phi_2\lceil_{\rho_1})\lceil_{\rho_2} = \phi_2\lceil_\rho = \phi\lceil_\rho,$$

where we again used the induction hypothesis.

When $\rho_2(x_j) = *$ and since (by induction) $\phi_1\lceil_\rho = (\phi_1\lceil_{\rho_1})\lceil_{\rho_2} = x_j^\epsilon$ when simplifying by ρ, ϕ_2 will also be simplified by the restriction $\rho^{(j)}$ and will be $(\phi_2\lceil_\rho)\lceil_{\rho(j)}$, which, by induction, is equal to $\phi_2\lceil_{\rho\circ\rho^{(j)}}$. On the other hand, when simplifying by ρ_1 and then by ρ_2, ϕ_2 reduces to $((\phi_2\lceil_{\rho_1})\lceil_{\rho^{(j)}})\lceil_{\rho_2}$ which, by applying the induction hypothesis twice, is equal to $\phi_2\lceil_{\rho_1\circ\rho^{(j)}\circ\rho_2}$. Since $\rho_1 \circ \rho^{(j)} \circ \rho_2 = \rho \circ \rho^{(j)}$, the lemma follows. □

We will need the above lemma in the case of analyzing what happens when we use the one-variable simplification rules. In that case the restriction ρ_2 will just give a non-$*$ value to one variable. Since the lemma is quite simple and natural, we will not always mention it explicitly when we use it.

Two examples to keep in mind during the proof are the following:

1. Suppose that ϕ computes the parity of m variables and is of size m^2. Then if p is small, the probability that ϕ will depend on exactly one variable is about $pm = p\sqrt{L(\phi)}$, and if p is large, we expect that the remaining formula will compute the parity of around pm variables and thus be of size at least $(pm)^2 = p^2 L(\phi)$.
2. Suppose that ϕ is the $\wedge$ of $L/2$ copies of $x_1 \vee x_2$. By our rules of simplification, this will not be simplified if both x_1 and x_2 are given the value $*$ by the restriction. Hence, with probability p^2 the entire formula remains and we get expected remaining size of at least $p^2 L$.

4. Reducing to size 1. We start by estimating the probability that a given formula reduces to size one. For notational convenience we set $q = \frac{2p}{1-p}$. This will be useful since we will change the values of restrictions at individual points and, if we change a non-$*$ value to $*$, we multiply the probability by q. Since we are interested in the case when p is small, q is essentially $2p$.

LEMMA 4.1. *Let ϕ be a formula of size L and ρ be a random restriction in R_p. Let E_δ be the event that ϕ is reduced to the constant δ by ρ for $\delta = 0, 1$. Furthermore let Δ be any filter. Then $Pr[L(\phi\lceil_\rho) = 1|\Delta]$ is bounded by*

$$q\left(LPr[E_0|\Delta]Pr[E_1|\Delta]\right)^{1/2}.$$

Remark. Most of the time the implied bound $q\sqrt{L/4}$ (which follows from $x(1-x) \leq 1/4$) will be sufficient.

Proof. Let ρ be a restriction that satisfies $L(\phi\lceil_\rho) = 1$ and belongs to Δ and suppose that ρ makes ϕ into the literal x_i^ϵ. By definition, $\rho(x_i) = *$ and we have two fellow restrictions ρ^δ, $\delta = 0, 1$, where ρ^δ is obtained by changing $\rho(x_i)$ to $xor(\epsilon, \delta)$. ρ^δ contributes to the event E_δ and by the definition of a filter belongs to Δ. Hence, we can identify the set of restrictions we are interested in with edges between restrictions that reduce the formula to the constants 0 and 1, respectively, and we are on familiar ground.

Let A be the set of restrictions that satisfy E_0 and belong to Δ, and let B be the set of restrictions that satisfy E_1 and belong to Δ. We partition $A \times B$ into rectangles $A_j \times B_j$, where for each j there is some variable x_{i_j} which takes a different value in A_j and B_j. This was first done in [10] (also see [7]), but here we will need a slight generalization and thus we choose to use the more intuitive framework introduced by Karchmer and Wigderson [4].

In the normal KW-game P_1 gets an input, x, from $f^{-1}(1)$ while P_0 gets an input, y, from $f^{-1}(0)$ and their task is to find a coordinate i such that $x_i \neq y_i$. This is solved by tracing the formula from the output to an input maintaining the property that the two inputs give different values to the gates on the path. This is achieved in the following way. At an $\wedge$-gate, P_0 points to the input of this gate that evaluates to 0 on y. Similarly, at an $\vee$-gate P_1 points to the input that evaluates to 1 on x.

We extend this game by giving P_δ a restriction ρ_δ that simplifies the formula to the constant δ. The task is to find an x_i on which the two restrictions take different values (we allow answers where one restriction takes the value $*$ and the other takes the value 0 or 1).

To solve this game, both players start by setting $\rho_\delta(x_j) = 1$ for each j such that $\rho_\delta(x_j) = *$. After this they play the standard KW-game. If the path ends at literal x_i^ϵ, then in the extended restrictions $\rho_\delta(x_i) = xor(\delta, \epsilon)$. Note that if $\rho_\delta(x_i) = 0$, then this was the initial value (since we only change values to 1), while if $\rho_\delta(x_i) = 1$ then the initial value was 1 or $*$. In either case we solve the problem.

The extended KW-game creates a partition of $A \times B$ and let $A_j \times B_j$ be the inputs that reach leaf j. Note that the fact that the set of inputs that reach a certain leaf is a product set follows from the fact that each move of the game is determined by one of the players based only on his own input. Let C_j be the set of restrictions ρ that satisfy $L(\phi\lceil_\rho) = 1$ and belong to Δ and such that the pair (ρ^0, ρ^1) reaches leaf j. By necessity the literal that appears at leaf j is the literal to which ρ reduced the formula. Now, note that the probability of C_j is bounded by q times the probability of A_j. This follows since the mapping $\rho \mapsto \rho^0$ gives a one-to-one correspondence of C_j with a subset of A_j and that $Pr(\rho) = qPr(\rho^0)$ for each ρ. We have the same relation between C_j and B_j; hence

$$Pr[L(\phi\lceil_\rho) = 1|\Delta] = \sum_j Pr[C_j|\Delta] \leq \sum_j q\left(Pr[A_j|\Delta]Pr[B_j|\Delta]\right)^{1/2}$$

$$\leq q\left(\sum_j 1\right)^{1/2}\left(\sum_j Pr[A_j|\Delta]Pr[B_j|\Delta]\right)^{1/2} = q\left(LPr[A|\Delta]Pr[B|\Delta]\right)^{1/2}$$

where we used the Cauchy–Schwarz inequality. □

Remark. Please note that the theorem of Khrapchenko is indeed a special case of this lemma. Khrapchenko starts with $A \subseteq f^{-1}(0)$, $B \subseteq f^{-1}(1)$, and C which is a set of edges of the form (a, b) where $a \in A$, $b \in B$, and the hamming distance between a and b is one. As noted above, each such edge naturally corresponds to a restriction ρ by setting $\rho(x_i) = a_i$ when $a_i = b_i$ and $\rho(x_{i_0}) = *$ for the unique coordinate i_0 such that $a_{i_0} \neq b_{i_0}$. Abusing notation, we can identify the restriction and the edge. Now setting $\Delta = A \cup B \cup C$ we get a filter, and since

$$\frac{Pr[C]}{q|C|} = \frac{Pr[A]}{|A|} = \frac{Pr[B]}{|B|},$$

the lemma reduces to Khrapchenko's theorem, i.e.,

$$L \geq \frac{|C|^2}{|A| \cdot |B|}.$$

5. The balanced case. In the general case we cannot hope to have an estimate which depends on the probability of reducing to either constant. The reason is because formulas that describe tautologies do not always reduce to the constant 1.

It remains to take care of the probability that the remaining formula is of size greater than 1.

DEFINITION 5.1. *Let $L^2(\phi)$ be the expected size of the remaining formula where we ignore the result if it is of size* 1*, i.e., $L^2(\phi) = \sum_{i=2}^{\infty} iPr[L(\phi\lceil_\rho) = i]$. Furthermore, let $L^2(\phi|\Delta)$ be the same quantity conditioned on $\rho \in \Delta$. Here we think of ρ as taken randomly from R_p and thus $L^2(\phi)$ depends on the parameter p. We will, however, suppress this dependence.*

To familiarize the reader with the ideas involved in the proof, we first prove the desired result when the formula is balanced. Here we will take the strictest possible definition of "balanced," namely that the formula is a complete binary tree, and just establish the size of the reduced formula as a function of its original depth.

THEOREM 5.2. *Let ϕ be a formula of depth d and ρ a random restriction in R_p. Let Δ be any filter and assume that $q \leq (d2^{1+d/2})^{-1}$; then*

$$L^2(\phi|\Delta) \leq q^2 d 2^{d-1}.$$

Proof. The proof is by induction over the depth. The base case ($d = 1$) is obvious. Now suppose $\phi = \phi_1 \wedge \phi_2$ (the $\vee$-case is similar), where the depth of each ϕ_i is $d - 1$ and hence the size is bounded by 2^{d-1}. We need to consider the following events:

1. $L(\phi_i\lceil_\rho) \geq 2$ for $i = 1$ or $i = 2$.
2. $L(\phi_1\lceil_\rho) = 1$, and the size of ϕ_2 after the simplification by ρ and the application of the one-variable simplification rule is at least 1.
3. $L(\phi_2\lceil_\rho) = 1$, and the size of ϕ_1 after the simplification by ρ and the application of the one-variable simplification rule is at least 1.

The estimate for $L^2(\phi|\Delta)$ is now

$$2 \cdot q^2(d-1)2^{d-2} + Pr[case\ 2] + Pr[case\ 3].$$

The first term comes from any subformula of size at least 2 appearing in either of the three cases, while the other two terms cover the new contribution in the respective cases.

Let us analyze the probability of case 2. Let p_i^ϵ be the probability that ϕ_1 reduces to x_i^ϵ. We know from Lemma 4.1 that

$$\sum p_i^\epsilon \leq q2^{(d-3)/2}.$$

Now consider the conditional probability that, given that ϕ_1 reduces to x_i^ϵ, ϕ_2 does not reduce to a constant. The condition that ϕ_1 reduces to x_i^ϵ can be written as "$\rho \in \Delta' \wedge \rho(x_i) = *$" for some filter Δ'. The reason for this is that if ρ reduces ϕ_1 to x_i^ϵ, then changing any $\rho(x_j)$, $j \neq i$ from $*$ to a constant the resulting restriction still reduces ϕ_1 to x_i^ϵ. This follows by Lemma 3.1. Thus we should work with the conditioning $\rho \in \Delta \bigcap \Delta' \wedge \rho(x_i) = *$. Now we substitute $x_i = \bar{\epsilon}$ in ϕ_2 and we can forget the variable x_i and just keep the restrictions in $\Delta \bigcap \Delta'$ that satisfy $\rho(x_i) = *$ (as restrictions on the other $n-1$ variables). This yields a filter Δ''. Thus we want to estimate the conditional probability that $\phi_2\lceil_{x_i=\epsilon}$ does not reduce to a constant given that $\rho \in \Delta''$. Now we are in position to apply induction, and hence this probability can be estimated by

$$q2^{(d-3)/2} + \frac{1}{2}q^2(d-1)2^{d-2},$$

where the first term is given by Lemma 4.1 and the second term is a bound on $\frac{1}{2}L^2(\phi_2|\Delta'')$. Using $q \leq (d2^{1+d/2})^{-1}$ we get the total bound $q2^{d/2-1}$. This implies that the total probability of case 2 is bounded by

$$\sum p_i^\epsilon q2^{d/2-1} \leq q^2 2^{d-5/2}.$$

The probability of case 3 can be bounded the same way and finally

$$2 \cdot q^2(d-1)2^{d-2} + q^2 2^{d-3/2} \leq q^2 d2^{d-1},$$

and the proof is complete. □

It is not difficult to extend Theorem 5.2 to larger d, but we leave the details to the reader.

6. The probability of size at least 2 remaining. The reason why things are so easy in the balanced case is that we need not rely on very complicated simplifications. In particular, we did not need the fact that a subformula can kill its brother. This will be needed in general, and we start with the following lemma.

LEMMA 6.1. *Let ϕ be a formula of size L and ρ be a random restriction in R_p. Let Δ be any filter, and assume that $q \leq (2\sqrt{L \log L})^{-1}$. Then the probability that $L(\phi\lceil_\rho) \geq 2$ conditioned on $\rho \in \Delta$ is at most $q^2 L(\log L)^{1/2}$.*

Proof. We prove the lemma by induction over the size of the formula. It is not hard to see that the lemma is correct when $L \leq 2$. Suppose that $\phi = \phi_1 \wedge \phi_2$ (the $\vee$-case is similar) where $L(\phi_i) = L_i$, $L_1 + L_2 = L$, and $L_1 \leq L_2$. We divide the event in question into the following pieces:

1. $L(\phi_1\lceil_\rho) \geq 2$.
2. $L(\phi_1\lceil_\rho) = 1$, and even after the one-variable simplification rule is used the size of the simplified ϕ_2 is at least 1.
3. ϕ_1 is reduced to the constant 1 by ρ and $L(\phi_2\lceil_\rho) \geq 2$.

The probability of the first event is by induction bounded by $q^2 L_1 (\log L_1)^{1/2}$. Suppose the probability, conditioned on $\rho \in \Delta$, that ϕ_1 is reduced to the constant 1 is Q. Call the corresponding set of restrictions Δ'. Note that Δ' is a filter. Then using the induction hypothesis with the conditioning $\Delta \bigcap \Delta'$ we get the probability of the third event to be bounded by

$$Q q^2 L_2 (\log L_2)^{1/2}.$$

Let us now estimate the probability of the second case. The probability that $L(\phi_1 \lceil_\rho) = 1$ is, by Lemma 4.1, bounded by $q \left(L_1 Q(1-Q)\right)^{1/2}$. To estimate the conditional probability that, given $L(\phi_1 \lceil_\rho) = 1$, ϕ_2 remains to be of size at least 1, we reason exactly as in the proof of Theorem 5.2. Hence, this probability can be estimated by

$$q\sqrt{L_2/4} + q^2 L_2 (\log L_2)^{1/2},$$

where the first term comes from an application of Lemma 4.1 and the second term from the inductive assumption. Thus, for the second case we get the total bound

$$q \left(L_1 Q(1-Q)\right)^{1/2} \times \left(q\sqrt{L_2/4} + q^2 L_2 (\log L_2)^{1/2} \right)$$

$$\leq q^2 \left(L_1 L_2 Q(1-Q)\right)^{1/2} \leq q^2 \left(L_1 L_2 (1-Q)\right)^{1/2},$$

where we used $q \leq (2\sqrt{L \log L})^{-1} \leq (2\sqrt{L_2 \log L_2})^{-1}$. We want to bound the sum of the estimates for probabilities of cases 2 and 3. Differentiating

$$Q L_2 (\log L_2)^{1/2} + \left(L_1 L_2 (1-Q)\right)^{1/2}$$

with respect to Q yields

$$L_2 (\log L_2)^{1/2} - \frac{1}{2} \left(\frac{L_1 L_2}{1-Q} \right)^{1/2}.$$

Thus the derivative is 0 when

$$(1-Q) = \frac{L_1}{4 L_2 \log L_2},$$

and this corresponds to a maximum since the second derivative is negative. Using this optimal value for Q we get a total estimate which is

$$q^2 L_1 (\log L_1)^{1/2} + q^2 L_2 (\log L_2)^{1/2} - \frac{q^2}{4} L_1 (\log L_2)^{-1/2} + \frac{q^2}{2} L_1 (\log L_2)^{-1/2}$$

$$= q^2 L_1 (\log L_1)^{1/2} + q^2 L_2 (\log L_2)^{1/2} + \frac{q^2}{4} L_1 (\log L_2)^{-1/2},$$

and we need to prove that this is less than $q^2 L (\log L)^{1/2}$ when $L_1 \leq L_2$ and $L = L_1 + L_2$. This is only calculus, but let us give the proof.

First note that $\log L_2 \geq \log \lceil L/2 \rceil \geq \frac{1}{2} \log L$ when $L \geq 3$; hence it is sufficient to bound

$$q^2 L_1 (\log L_1)^{1/2} + q^2 L_2 (\log L_2)^{1/2} + \frac{q^2}{2} L_1 (\log L)^{-1/2}.$$

Now if we set $H(x) = x(\log x)^{1/2}$ it is not difficult to see that $H''(x)$ is positive for $x \geq 2$ and hence the above expression is convex for $L_1 \geq 2$. This means that it is maximized either for $L_1 = 1, 2$ or $L_1 = L_2$. The first two cases correspond to the inequalities

$$(L-1)(\log(L-1))^{1/2} + \frac{1}{2}(\log L)^{-1/2} \leq L(\log L)^{1/2}$$

and

$$2 + (L-2)(\log(L-2))^{1/2} + (\log L)^{-1/2} \leq L(\log L)^{1/2}$$

which are easily seen to hold for $L \geq 3$ and $L \geq 4$, respectively. To check the required inequality when $L_1 = L_2$ we need to prove that

$$L(\log L/2)^{1/2} + \frac{L}{4}(\log L)^{-1/2} \leq L(\log L)^{1/2}.$$

This follows from $\sqrt{x} - \sqrt{x-1} \geq \frac{1}{2\sqrt{x}}$, which is valid for all $x \geq 1$. □

7. Main shrinkage theorem. We are now ready for our main theorem.

THEOREM 7.1. *Let ϕ be a formula of size L and ρ be a random restriction in R_p. Then the expected size of $\phi\lceil_\rho$ is bounded by*

$$O\left(p^2\left(1 + \left(\log\left(\min\left(\frac{1}{p}, L\right)\right)\right)^{3/2}\right)L + p\sqrt{L}\right).$$

Crucial to this theorem is the following lemma.

LEMMA 7.2. *Let ϕ be a formula of size L and ρ be a random restriction in R_p. If $q \leq (2\sqrt{L\log L})^{-1}$, then*

$$L^2(\phi) \leq 30q^2L(\log L)^{3/2},$$

while if $\frac{1}{2} \geq q \geq (4\sqrt{L\log L})^{-1}$, then

$$L^2(\phi) \leq 200q^2L(\log q^{-1})^{3/2}.$$

First note that Theorem 7.1 follows from Lemma 7.2 together with Lemma 4.1; thus it is sufficient to prove Lemma 7.2. Also note that Lemma 7.2 and Theorem 7.1 do not consider conditional probabilities. There are two reasons for this. It is not needed and the proof does not seem to allow it.

Proof of Lemma 7.2. The hard part of the lemma is the case when q is small. We will start by establishing this case. As before we proceed by induction. The base case ($L = 1$) is obvious. Suppose that $\phi = \phi_1 \wedge \phi_2$ (the $\vee$-case is similar), where $L(\phi_i) = L_i$, $L_1 + L_2 = L$, and $L_1 \leq L_2$.

Our basic estimate for $L^2(\phi)$ is $L^2(\phi_1) + L^2(\phi_2)$, which by induction can be bounded by

$$S(L_1, L_2) = 30q^2L_1(\log L_1)^{3/2} + 30q^2L_2(\log L_2)^{3/2}.$$

We need to revise this estimate, and in particular, we need to consider the events which contribute either to the event described by the basic estimate ($L(\phi_i\lceil_\rho) \geq 2$ for $i = 1$, or $i = 2$) or to the event we are trying to estimate ($L(\phi\lceil_\rho) \geq 2$):

1. We have $L(\phi_i\lceil_\rho) \geq 2$ for $i = 1$ and $i = 2$.
2. $L(\phi_i\lceil_\rho) = 1$ for $i = 1$ or $i = 2$, and the one-variable simplification rule was active at the top gate.
3. $L(\phi_1\lceil_\rho) = L(\phi_2\lceil_\rho) = 1$, and the one-variable simplification rule was not active at the top gate.
4. $L(\phi_1\lceil_\rho) = 1$, $L(\phi_2\lceil_\rho) \geq 2$, and the one-variable simplification rule was not active at the top gate.
5. $L(\phi_2\lceil_\rho) = 1$, $L(\phi_1\lceil_\rho) \geq 2$, and the one-variable simplification rule was not active at the top gate.
6. The function ϕ_1 is reduced to the constant 0 while $L(\phi_2\lceil_\rho) \geq 2$.
7. The function ϕ_1 is reduced to the constant 1 while $L(\phi_2\lceil_\rho) \geq 2$.
8. The function ϕ_2 is reduced to the constant 0 while $L(\phi_1\lceil_\rho) \geq 2$.
9. The function ϕ_2 is reduced to the constant 1 while $L(\phi_1\lceil_\rho) \geq 2$.

Let us first investigate what corrections we need to make to our basic estimate in the various cases.

Case 1. The basic estimate is correct.

Case 2. Suppose that ϕ_1 reduces to x_i^ϵ. If the resulting formula is of size at least 2, then ϕ_2 was of size at least 2 before we did the simplification of substituting $\bar{\epsilon}$ for x_i. This reduced the size of ϕ_2 by at least one. This means that the formula size of ϕ is at most the formula size of ϕ_2 before we did this simplification. However, in our basic estimate we have not taken this simplification into account, and thus we need not add anything to our basic estimate in this case.

Case 3. In this case we need to add 2 to our basic estimate. We will need to do some work to estimate the probability of this case.

Case 4. In this case we need to add 1 to our basic estimate. Also, the probability of this event needs a little bit of work.

Case 5. In this case we need to add 1 to our basic estimate. From our previous work we can estimate the probability of this event simply by the probability that the remaining size of ϕ_1 is at least 2 and that by Lemma 6.1 this probability is bounded by

$$q^2 L_1 (\log L_1)^{1/2}.$$

Cases 6 *and* 8. In these cases we can subtract at least 2 from our original estimate. This follows since in these cases we erase a formula of size at least 2 which contributed needlessly to the basic estimate. We will use only Case 6.

Cases 7 *and* 9. The basic estimate is correct.

The above reasoning gives the total bound

$$S(L_1, L_2) + 2Pr[case\ 3] + Pr[case\ 4] + Pr[case\ 5] - 2Pr[case\ 6].$$

We have already bounded $Pr[case\ 5]$, and for the other probabilities we will establish the following lemmas.

LEMMA 7.3. *If* $q \leq (2\sqrt{L_1 \log L_1})^{-1}$, *then*

$$Pr[case\ 3] \leq 20q^2 L_1 + \frac{1}{2} q^2 L_1 (\log L_1)^{1/2} + \frac{1}{2} Pr[case\ 6].$$

LEMMA 7.4. *If* $q \leq (2\sqrt{L_1 \log L_1})^{-1}$, *then*

$$Pr[case\ 4] \leq q^2 L_1 + Pr[case\ 6].$$

Let us check that this is sufficient to prove Lemma 7.2 in the case where q is small.

$$S(L_1, L_2) + 2Pr[case\ 3] + Pr[case\ 4] + Pr[case\ 5] - 2Pr[case\ 6]$$

$$\leq S(L_1, L_2) + 41q^2L_1 + q^2L_1(\log L_1)^{1/2}.$$

We need to prove that this is bounded by $30q^2L(\log L)^{3/2}$ for all possible L_1. Substituting $L_2 = L - L_1$ and differentiating twice we see that this is a convex function of L_1 when $2 \leq L_1 \leq L/2$. This means that it is maximized either for $L_1 = 1, 2$ or $L_1 = L_2$.

For $L_1 = 1$ we need to establish

$$30(L-1)(\log(L-1))^{3/2} + 41 \leq 30L(\log L)^{3/2},$$

which is easily checked to be true for $L = 2$ and $L = 3$. For $L \geq 4$ the inequality follows from

$$L(\log L)^{3/2} \geq (L-1)(\log L)^{3/2} + 2.$$

For $L_1 = 2$ we need to check

$$30(L-2)(\log(L-2))^{3/2} + 60 + 82 + 1 \leq 30L(\log L)^{3/2},$$

which for $L \geq 4$ follows from

$$L(\log L)^{3/2} \geq (L-2)(\log L)^{3/2} + 2 \cdot 2^{3/2}$$

and $60 \cdot 2^{3/2} > 143$.

Finally, for $L_1 = L/2$ we need to estimate

$$30L(\log L/2)^{3/2} + L/2(41 + (\log L/2)^{1/2}),$$

and using the inequality $x^{3/2} - (x-1)^{3/2} \geq x^{1/2}$, which is valid for any $x \geq 1$, this can be bounded by

$$30L(\log L)^{3/2} - 30L(\log L)^{1/2} + \frac{43}{2}L(\log L)^{1/2} \leq 30L(\log L)^{3/2}.$$

Thus we are done.

Hence we need only establish Lemmas 7.3 and 7.4. The basic principle is to start with a set of restrictions that contributes to the bad case (cases 3 and 4, respectively) and create a set of restrictions that contribute to the good case, namely case 6. In this process there will be some "spills" and hence we need the additive terms. Lemma 7.4 is by far easier, and since the basic outline is the same, we start with this lemma.

Proof of Lemma 7.4. Let C be the set of restrictions such that ϕ_1 reduces to exactly one variable or its negation and such that the reduced ϕ_2 does not contain this variable. Let A be the set of restrictions that is formed by setting the variable that remains in ϕ_1 in such a way to make ϕ_1 reduce to the constant 0, and let B be the corresponding set that makes ϕ_1 reduce to 1. Each element in C corresponds to an edge between A and B and we can (as in the proof of Lemma 4.1) let this define a path in ϕ_1. Thus each leaf in ϕ_1 corresponds to a set $A_j \times B_j$ which reaches this leaf and a subset C_j of C such that for any $\rho \in C_j$, its neighbors belong to A_j and B_j,

respectively. The sets $A_j \times B_j$ form a partition of $A \times B$. Further suppose that the literal at leaf j of ϕ_1 is $x_{d_j}^{\epsilon_j}$. Note that this implies that if $\rho \in C_j$, then ρ simplifies ϕ_1 to $x_{d_j}^{\epsilon_j}$.

Let q_j be the conditional probability that, when ρ is chosen uniformly from C_j, $L(\phi_2\lceil_\rho) \geq 2$. The probability of case 4 is then given by

$$\sum_j Pr[C_j]q_j.$$

If we take any restriction ρ contributing to this event and change the value of ρ at x_{d_j} to ϵ_j then we get a restriction ρ' contributing to Case 6. This follows since x_{d_j} does not appear in the reduced ϕ_2. The set of restrictions created at leaf j will be of total probability $q^{-1}Pr[C_j]q_j$ and we seem to be in good shape. However, the same restriction ρ' might be created at many leaves and hence we would be overcounting if we would just sum these probabilities for various j. In addition, note that ρ' belongs to A, and if it is created at leaf j, then it belongs to A_j. Now, since $A_j \times B_j$ forms a partition of $A \times B$, we have for any $\rho \in A$

$$\sum_{j|\rho \in A_j} Pr[B_j] = Pr[B] \leq 1.$$

This means that if we multiply the total probability of restrictions created at leaf j by $Pr[B_j]$, we avoid overcounting. Thus the sure contribution to the probability of Case 6 is

$$\sum_j q^{-1}Pr[C_j]Pr[B_j]q_j.$$

We need to compare this to $\sum_j Pr[C_j]q_j$. For the j for which $Pr[B_j] \geq q$, the term in the sum for Case 6 is bigger than the corresponding term for Case 4, while for other j, we use that $Pr[C_j] \leq qPr[B_j] \leq q^2$; thus summing over those j gives a contribution of at most q^2L_1. We have proved Lemma 7.4. □

Next we turn to Lemma 7.3. This will be more complicated, mainly because the restrictions contributing to Case 6 are more difficult to construct.

Proof of Lemma 7.3. Let A_j, B_j, and C_j be as in the previous proof. For fixed j let r_j be the conditional probability that $L(\phi_2\lceil_\rho) = 1$ given that $\rho \in C_j$. We divide the leaves into two cases depending on whether $r_j \leq 20qPr[B_j]^{-1}$. If we restrict the summation to those j that satisfy this inequality, then

$$\sum_j Pr[C_j]r_j \leq \sum_j qPr[B_j]20qPr[B_j]^{-1} \leq 20q^2L_1$$

and this gives the first term on the right-hand side of Lemma 7.3. We thus concentrate on the case when $r_j \geq 20qPr[B_j]^{-1}$.

Let $A^{2,j}$ and $B^{2,j}$ be the subsets of C_j which reduce ϕ_2 to 0 and 1, respectively. Let ρ be a restriction that belongs to C_j and contributes to Case 3. Assume that ρ reduces ϕ_2 to x_d^ϵ. We can obtain a restriction ρ' in $A^{2,j}$ (if $\epsilon = 1$) or $B^{2,j}$ (if $\epsilon = 0$) by setting $\rho'(x_k) = \rho(x_k)$ for $k \neq d$ and $\rho'(x_d) = 1$. To see that $\rho' \in C_j$, note that before we play the KW-game we give all variables given the value $*$ by ρ the value 1. Thus the executions on ρ and ρ' are identical and $\rho' \in C_j$. Also, clearly ρ' forces ϕ_2 to $\bar{\epsilon}$. Now suppose that

$$\sum_{\rho|\rho' \in A^{2,j}} Pr[\rho] \geq \sum_{\rho|\rho' \in B^{2,j}} Pr[\rho] \tag{2}$$

(the other case being symmetric). Suppose that $A^{2,j}$ consists of the restrictions $\rho_1 \dots \rho_k$. For ρ_i we define a set of *critical variables*, and x_k is in this set if

- $\rho_i(x_k) = 1$.
- creating the restriction ρ_i' by setting $\rho_i'(x_l) = \rho_i(x_l)$ for every $l \neq k$ while $\rho_i'(x_k) = *$ creates a restriction in C and $\phi_2 \lceil_{\rho_i'}$ reduces to $\bar{x}_k$.

Note that, as observed above, $\rho' \in C$ in fact implies that $\rho' \in C_j$ since $\rho \in C_j$, and we go from ρ' to ρ by changing the value on a variable from $*$ to 1.

Suppose that there are s_i critical variables for ρ_i. By definition, each restriction contributing to the conditional probability r_j gives one critical variable for one $\rho_i \in A^{2,j}$ exactly when ϕ_2 is reduced to a negated variable; otherwise it gives no contribution. By (2) the first case happens in at least half the cases and hence we have

$$r_j = \alpha Pr[C_j]^{-1} \sum_i q s_i Pr[\rho_i]$$

for some α satisfying $1 \leq \alpha \leq 2$. We now create a set of restrictions in the following way. We obtain $\binom{s_i}{2}$ new restrictions from ρ_i by choosing two critical variables for ρ_i, and for each such choice (s,t) create a restriction $\rho_i^{(s,t)}$ by setting $\rho_i^{(s,t)}(x_k) = \rho_i(x_k)$ for every $k \notin \{s,t\}$ while $\rho_i^{(s,t)}(x_s) = \rho_i^{(s,t)}(x_t) = *$. This way, we get a set of restrictions of total probability $q^2\binom{s_i}{2}Pr[\rho_i]$.

Let us relate the total probability of these constructed restrictions to r_j. Note that

$$\begin{aligned} r_j^2 &= \left(\alpha Pr[C_j]^{-1} \sum_i (q s_i Pr[\rho_i])\right)^2 \\ &\leq \alpha^2 \left(Pr[C_j]^{-1} \sum_i (q^2 s_i^2 Pr[\rho_i])\right) \cdot \left(Pr[C_j]^{-1} \sum_i Pr[\rho_i]\right) \\ &= \alpha^2 Pr[C_j]^{-1} Pr[A^{2,j}|C_j] \sum_i q^2 s_i^2 Pr[\rho_i], \end{aligned}$$

where the inequality comes from the Cauchy–Schwarz inequality. Now

$$\begin{aligned} \sum_i q^2 \binom{s_i}{2} Pr[\rho_i] &= \sum_i \frac{1}{2} q^2 s_i^2 Pr[\rho_i] - \sum_i \frac{1}{2} q^2 s_i Pr[\rho_i] \\ &\geq \frac{r_j^2 Pr[C_j]}{2\alpha^2 Pr[A^{2,j}|C_j]} - \frac{q}{2\alpha} r_j Pr[C_j] \\ &\geq \left(\frac{1}{2\alpha^2} - \frac{1}{40\alpha}\right) r_j^2 Pr[C_j] \geq \frac{1}{10} r_j^2 Pr[C_j], \end{aligned}$$

since $\alpha \leq 2$ and $r_j \geq 20q$.

Remark. Note that the constructed restrictions need not be in C_j. The reason is that there is no control when a variable is changed from being fixed to being $*$. In particular, if we were trying to estimate a conditional expectation, we would be in deep trouble, since it need not be the case that these recently constructed restrictions satisfy the condition.

Let us look more closely at these obtained restrictions. They give the value $*$ to the variable x_{d_j} since the restriction that we started with belonged to C_j. They also give the value $*$ to the two special variables x_s and x_t.

We now change the value at x_{d_j} to ϵ_j in an attempt to force ϕ_1 to 0. Note that this attempt might not always be successful since, once x_s and x_t become unassigned, ϕ_1 might also depend on those variables (as well as on others). We leave this problem for the time being. Let us analyze the set of restrictions created in this way.

At leaf j we created a set of restrictions of total probability at least $q^{-1}\frac{1}{10}Pr[C_j]r_j^2$. However, the same restriction might appear many times and we need to adjust for this fact. Take any restriction ρ created from $\rho_i \in A^{2,j}$. First note that ρ determines the identity of the two special variables x_s and x_t. These are, namely, the only variables x_k given the value $*$ by ρ with the property that setting $\rho(x_k) = 1$ makes ϕ_2 depend on only one variable. This follows since we recreate a restriction from C_j with the additional property that x_{d_j} is set, but since we are considering cases when ϕ_2 was independent of x_{d_j}, setting a value to x_{d_j} does not matter. To complete the characterization of x_s and x_t, note that after setting any other variable x_k to any value it is still true that ϕ_2 depends on both x_s and x_t.

Let x_s be the variable with lower index of the variables x_s and x_t which we just have identified. Consider the restriction ρ' obtained by setting $\rho'(x_s) = 1$ while $\rho'(x_k) = \rho(x_k)$ for every $x_k \neq x_s$. We claim that ρ' belongs to A_j. Remember that $A_j \times B_j$ was the set of inputs reaching leaf j when playing the KW-game on the formula ϕ_1. To see this claim let ρ'' be obtained by setting $\rho''(x_k) = \rho'(x_k)$ for $x_k \neq x_{d_j}$ while $\rho''(x_{d_j}) = *$. By the conditions for x_t being a critical variable for ρ_i, $\rho'' \in C_j$ and hence $\rho' \in A_j$.

Thus, starting with ρ we have created a unique restriction ρ' such that whenever ρ is created at leaf j then $\rho' \in A_j$. Thus, reasoning as in the proof of Lemma 7.4, if we multiply the probability of the restrictions produced at leaf j by $Pr[B_j]$, then we avoid making an overestimate. This means that we have created a set of restrictions of total probability at least

$$\sum_j \frac{1}{10} q^{-1} Pr[C_j] r_j^2 Pr[B_j].$$

The created restrictions are of two types: either they do reduce the formula ϕ_1 to 0 or they do not. In the former case they contribute to Case 6 (since ϕ_2 depends on x_s and x_t), and we have to estimate the probability of the latter case. We claim that in this case the reduced ϕ_1 contains both the variables x_s and x_t. This follows, since setting $\rho(x_s) = 1$ or $\rho(x_t) = 1$ simplifies ϕ_1 to 0 which in its turn is basically the fact $\rho' \in A_j$ established above. By Lemma 6.1 it follows that the probability of this case is bounded by $q^2 L_1 (\log L_1)^{1/2}$. Summing up we have

$$\begin{aligned} Pr[case\ 3] = \sum_{j=1}^{L_1} Pr[C_j] r_j &= \sum_{j|r_j\ small} Pr[C_j] r_j + \sum_{j|r_j\ large} Pr[C_j] r_j \\ &\leq 20 q^2 L_1 + \frac{1}{20} \sum_j q^{-1} Pr[C_j] r_j^2 Pr[B_j] \\ &\leq 20 q^2 L_1 + \frac{1}{2} \left(Pr[case\ 6] + q^2 L_1 (\log L_1)^{1/2} \right), \end{aligned}$$

where the first inequality uses the bound for r_j and the last inequality is based on the above reasoning. The proof of Lemma 7.3 is complete. □

All that remains is to complete the proof of Lemma 7.2 when $q \geq (4\sqrt{L \log L})^{-1}$. To simplify the calculations we will, in this case, prove the slightly stronger bound

$$L^2(\phi) \leq 200 q^2 L (\log q^{-1})^{3/2} - 2.$$

First note that when $(4\sqrt{L\log L})^{-1} \le q \le (2\sqrt{L\log L})^{-1}$, the second bound follows from the first bound since

$$\begin{aligned}200q^2L(\log q^{-1})^{3/2} &\ge 200q^2L(1/2\log L)^{3/2} \ge 62q^2L(\log L)^{3/2}\\ &\ge 30q^2L(\log L)^{3/2} + 32(4\sqrt{L\log L})^{-2}L(\log L)^{3/2}\\ &\ge 30q^2L(\log L)^{3/2} + 2.\end{aligned}$$

It remains to establish the second bound when $q \ge (2\sqrt{L\log L})^{-1}$, and we do this by induction over L. Assume that $\phi = \phi_1 \wedge \phi_2$, (the $\vee$-case being similar) where $L(\phi_i) = L_i$, $L_1 + L_2 = L$, and $L_1 \le L_2$. This implies that we can always use the second bound when bounding $L^2(\phi_2)$. We have two cases, depending on whether or not $q \le (4\sqrt{L_1\log L_1})^{-1}$. If $q \ge (4\sqrt{L_1\log L_1})^{-1}$ then, using the induction hypothesis and $L^2(\phi) \le L^2(\phi_1) + L^2(\phi_2) + 2$, the result follows immediately.

To take care of the other case, notice that our estimates for the corrections to the basic estimates depended only on L_1. This means that in this case we get the total bound

$$L^2(\phi_1) + L^2(\phi_2) + 41q^2L_1 + q^2L_1(\log L_1)^{1/2},$$

and using the induction hypothesis (the first case for ϕ_1 and the second for ϕ_2) we can bound this by

$$\begin{aligned}&30q^2L_1(\log L_1)^{3/2} + (200q^2L_2(\log q^{-1})^{3/2} - 2) + q^2L_1(41 + (\log L_1)^{1/2})\\ &\quad\le 90q^2L_1(\log q^{-1})^{3/2} + (200q^2L_2(\log q^{-1})^{3/2} - 2) + 43q^2L_1(\log q^{-1})^{1/2}\\ &\quad\le 200q^2L(\log q^{-1})^{3/2} - 2,\end{aligned}$$

and the proof is complete. □

8. Application to formula-size lower bounds. As mentioned in the introduction, it is well known that shrinkage results can be used to derive lower bounds on formula size. Let us briefly recall the function which seems to be the most appropriate for this purpose. The input bits are of two types. For notational simplicity assume that we have $2n$ input bits and that $\log n$ is an integer that divides n. The first n bits define a Boolean function H on $\log n$ bits. The other n bits are divided into $\log n$ groups of $n/\log n$ bits each. If the parity of the variables in group i is y_i then the output is $H(y)$. We call this function A as it was first defined by Andreev [9].

THEOREM 8.1. *The function A requires formulas of size*

$$\Omega\left(\frac{n^3}{(\log n)^{7/2}(\log\log n)^3}\right).$$

Proof. Assume that we have a formula of size S which describes A. We know ([8, Chapter 4, Theorem 3.1]) that there is a function of $\log n$ variables which requires a formula size which is

$$\Omega\left(\frac{n}{\log\log n}\right).$$

We fix the first set of values to describe such a function. This might decrease the size of the formula, but it is not clear by how much, and hence we just note that the resulting formula is of size at most S. Apply an R_p-restriction with $p = \frac{2\log n\log\log n}{n}$ on the

remaining formula. By our main theorem the resulting formula will be of expected size at most $O(Sn^{-2}(\log n)^{7/2}(\log\log n)^2+1)$. The probability that all variables in a particular group are fixed is bounded by

$$(1-p)^{\frac{n}{\log n}} \le e^{-\frac{pn}{\log n}} \le (\log n)^{-2}.$$

Since there are only $\log n$ groups, with probability $1-o(1)$ there remains at least one live variable in each group. Now since a positive random variable is at most twice its expected size with probability at least $1/2$, it follows that there is a positive probability that we have at most twice the expected remaining size and some live variable in each group. It follows that

$$O\left(Sn^{-2}(\log n)^{7/2}(\log\log n)^2\right) \ge \Omega\left(\frac{n}{\log\log n}\right),$$

and the proof is complete. □

We note that there are indeed formulas for the function A of size $O(n^3(\log n)^{-1})$, and hence our bounds are close to optimal.

9. Conclusions. There remain two interesting questions on shrinking:

- *What is the shrinkage exponent for monotone formulas*? In some sense we have established that it is 2; namely, one of the two examples given in the introduction is monotone and shrinks only by a factor p^2. This is the example of $L/2$ copies of $x_1 \wedge x_2$. This is not a natural example, and if it is the only one, we are asking the wrong question. We can get around this example by using 2-variable and 3-variable simplification rules. We could also ask a slightly different question, namely, what is the minimal α such that for arbitrary small p there is a monotone formula of size $O(p^{-\alpha})$ that is trivialized by a restriction from R_p with probability at most $1/2$?

  Apart from its inherent interest, a successful answer to this question would in most cases (depending on the exact form of the answer) lead to an $\omega(n^2)$ lower bound for monotone formulas for the majority function.
- *Are these annoying log factors really needed*? This is really of minor importance. If they were indeed needed it would be surprising.

Acknowledgments. I am most grateful to Sasha Razborov and V.N. Lebedev for catching gaps in the proofs in previous versions of this paper. Together with Andy Yao, Sasha Razborov was also very helpful in giving ideas in the initial stages of this research. Finally, I am most grateful to two anonymous referees for their careful reading of the manuscript.

REFERENCES

[1] M. Dubiner and U. Zwick, *How do read-once formulae shrink*?, Combin. Prob. Comput., 3 (1994), pp. 455–469.

[2] M. Furst, J. B. Saxe, and M. Sipser, *Parity, circuits and the polynomial time hierarchy*, Math. Systems Theory, 17 (1984), pp. 13–27.

[3] J. Håstad, A. Razborov, and A. Yao, *On the shrinkage exponent for read-once formulae*, Theoret. Comput. Sci., 141 (1995), pp. 269–282.

[4] M. Karchmer and A. Wigderson, *Monotone circuits for connectivity require super-logarithmic depth*, SIAM J. Discrete Math., 3 (1990), pp. 255–265.

[5] N. Nisan and R. Impagliazzo, *The effect of random restrictions on formulae size*, Random Structures Algorithms, 4 (1993), pp. 121–134.

[6] M. S. Paterson and U. Zwick, *Shrinkage of de Morgan formulae under restriction*, Random Structures Algorithms, 4 (1993), pp. 135–150.
[7] A. Razborov, *Applications of matrix methods to the theory of lower bounds in computational complexity,* Combinatorica, 10 (1990), pp. 81–93.
[8] I. Wegener, *The Complexity of Boolean Function*, John Wiley, New York, 1987.
[9] A. E. Andreev, *On a method for obtaining more than quadratic effective lower bounds for the complexity of π-schemes*, Moscow Univ. Math. Bull., 42 (1987), pp. 63–66 (in Russian).
[10] K. L. Rychkov, *A modification of Khrapchenko's method and its application to bounding the complexity of Π-networks for coding functions*, in Methods of Discrete Analysis in the Theory of Graphs and Circuits, Institut Matematiki SOAN SSSR, Novosibirsk, 1985, pp. 91–98 (in Russian).
[11] B. A. Subbotovskaya, *Realizations of linear functions by formulas using $+, *, -$*, Soviet Math. Dokl., 2 (1961), pp. 110–112 (in Russian).
[12] V. M. Khrapchenko, *Complexity of the realization of a linear function in the class of π-circuits*, Math. Notes Acad. Sci. USSR, 9 (1971), pp. 21–23 (in Russian).
[13] V. M. Khrapchenko, *A method of determining lower bounds for the complexity of Π-schemes*, Math. Notes Acad. Sci. USSR, 10 (1971), pp. 474–479 (in Russian).

SIAM J. COMPUT.
Vol. 27, No. 1, pp. 65–89, February 1998
© 1998 Society for Industrial and Applied Mathematics
003

SHARED MEMORY CONSISTENCY CONDITIONS FOR NONSEQUENTIAL EXECUTION: DEFINITIONS AND PROGRAMMING STRATEGIES*

HAGIT ATTIYA†, SOMA CHAUDHURI‡, ROY FRIEDMAN§, AND JENNIFER L. WELCH¶

Abstract. To enhance performance on shared memory multiprocessors, various techniques have been proposed to reduce the latency of memory accesses, including pipelining of accesses, out-of-order execution of accesses, and branch prediction with speculative execution. These optimizations can, however, complicate the user's model of memory. This paper attacks the problem of simplifying programming on two fronts.

First, a general framework is presented for defining shared memory consistency conditions that allows nonsequential execution of memory accesses. The interface at which conditions are defined is between the program and the system and is architecture-independent. The framework is used to generalize three consistency conditions—sequential consistency, hybrid consistency, and weak consistency—for nonsequential execution. Thus, familiar consistency conditions can be precisely specified even in optimized architectures.

Second, three techniques are described for structuring programs so that a shared memory that provides the weaker (and more efficient) condition of hybrid consistency appears to guarantee the stronger (and more costly) condition of sequential consistency. The benefit of these techniques is that sequentially consistent executions are easier to reason about. The first technique statically classifies accesses based on their type. This approach is extremely simple to use and leads to a general technique for writing efficient synchronization code. The third technique is to avoid data races in the program, which was previously studied in a somewhat different setting.

Precise, yet short and comprehensible, proofs are provided for the correctness of the programming techniques. Such proofs shed light on the reasons these techniques work; we believe that the insight gained can lead to the development of other techniques.

Key words. distributed shared memory, consistency conditions, sequential consistency

AMS subject classifications. 68-02, 68M07, 68Q10, 68Q60, 68Q65,

PII. S0097539794278396

1. Introduction.

1.1. Overview. Shared memory multiprocessors are an interesting alternative to message-passing distributed computer systems. The parallelism in multiprocessors

* Received by the editors December 9, 1994; accepted for publication (in revised form) December 18, 1995. An extended abstract of this paper appeared in the *Proceedings of the 5th ACM Symposium on Parallel Algorithms and Architectures,* ACM, New York, 1993, pp. 241–250. This work was supported by United States–Israel Binational Science Foundation (BSF) grant 92-0233; Technion V.P.R. Argentinian Research Fund for the promotion of research in the Technion; NSF grants CCR-89-15206, CCR-9010730, and CCR-9308103; DARPA contracts N00014-89-J-1988 and N00014-92-J-4033; ONR contract N00014-91-J-1046; a grant from the ISU Graduate College; an IBM Faculty Development Award; NSF Presidential Young Investigator Award CCR-9158478; and TAMU Engineering Excellence funds.

http://www.siam.org/journals/sicomp/27-1/27839.html

† Department of Computer Science, The Technion, Haifa 32000, Israel (hagit@cs.technion.ac.il).

‡ Department of Computer Science, Iowa State University, Ames, IA 50011 (chaudhur@cs.iastate.edu). Much of this research was completed while this author was with the Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA.

§ Department of Computer Science, Cornell University, Ithaca, NY 14853 (roy@cs.cornell.edu). This research was completed while the author was with the Department of Computer Science, Technion, Haifa 32000, Israel.

¶ Department of Computer Science, Texas A&M University, College Station, TX 77843-3112 (welch@cs.tamu.edu). Much of this research was completed while this author was with the Department of Computer Science, University of North Carolina, Chapel Hill, NC.

offers the potential of greatly increased performance, and shared memory is an attractive communication paradigm. Unfortunately, access to shared memory locations is a major bottleneck for the performance of multiprocessors. The high latency of memory operations is due to the amount of time needed to execute memory operations locally and to the interprocessor communication delay, which increases with the number of processors.

Many uniprocessor architecture techniques have been developed and implemented to hide the latency of memory operations by allowing operations to overlap. These techniques include performing memory accesses in parallel, pipelining memory accesses, initiating accesses out of order, and speculative execution (in which the system predicts the outcome of future conditional branches and begins memory accesses for the predicted branch). Specific instances include [1, 20, 24, 34, 35, 37, 39, 40]. Since these techniques deviate from the sequential order of memory accesses specified by the program, we call them "nonsequential."

In multiprocessors, memory is often replicated and distributed to compensate for the interprocessor communication delay. A consistency condition for shared memory specifies what guarantees are provided about the values returned in the presence of concurrent accesses to different copies of the memory. A variety of consistency conditions have previously been proposed for shared memory architectures, e.g., [3, 4, 8, 9, 12, 18, 19, 21, 23, 25, 28, 29]. Clearly, the consistency condition limits the optimizations that can be implemented.

A natural question is how to define consistency conditions that allow nonsequential execution of memory accesses, in a manner that admits optimized systems. A further question is how to program these optimized systems with these generalized consistency conditions in a safe and effective manner.

In this paper, we focus on the interface between programs and the shared memory under consistency conditions that allow nonsequential execution. We present a framework for defining such conditions as properties of executions, rather than as properties of hardware implementations. We hope that this aspect of our framework will elucidate these somewhat complex conditions for algorithm designers, programmers, and perhaps compiler designers. We demonstrate our framework by extending three consistency conditions to allow for nonsequential execution of memory accesses. For one particular condition that allows efficient implementations, we present and prove the correctness of three programming techniques that provide the illusion of a stronger and more natural condition. The first approach is based on a static labeling of the memory accesses in the program according to their type. The second approach is to access shared data only inside appropriately protected critical sections. The third approach is to avoid data races, an approach pioneered in [3, 4, 22, 23] for hardware implementations.

1.2. Detailed description. Our first contribution is a framework for defining consistency conditions that is general enough to allow nonsequential execution of memory operations. This framework combines the following two features: (1) The interface at which conditions are specified in our model is between the program and the system instead of being specified as the internal behavior of the system. This is the interface used in [12, 25, 28] and, in our opinion, is the natural one to use for specification to be independent of implementation. (2) The framework allows for arbitrary optimizations by the system, including especially speculative execution of memory accesses. Recent experiments have shown that if the program has complex control flow then only moderate speedup can be achieved by parallelism without speculative execution (e.g., [26, 41]).

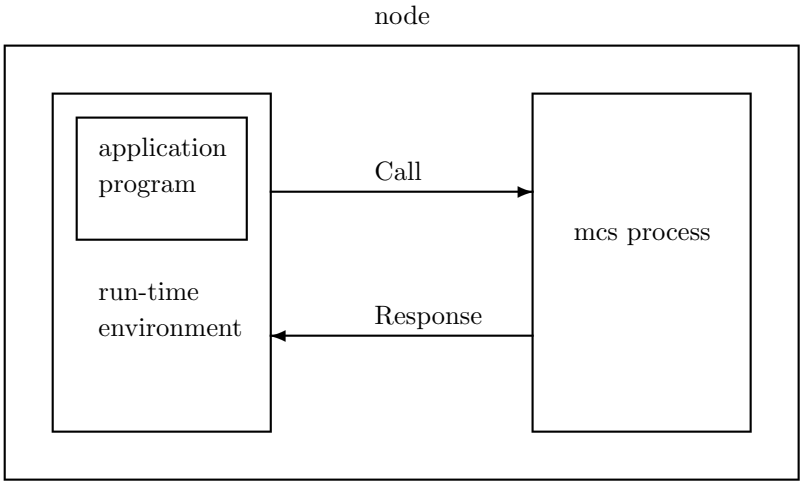


FIG. 1. *A node.*

The framework is then used to extend three known consistency conditions for nonsequential execution. Our extensions have two pleasing properties: (1) The conditions are defined for all programs, not just programs that satisfy certain conditions. (2) We give a formal, yet intuitive, treatment of explicit control instructions, which are crucial for expressing the flow of control in a program and in analyzing its correctness on nonsequential implementations.

Our framework assumes a system consisting of a collection of nodes. At each node there is an *application program*, a *memory consistency system* (mcs) process, and a *run-time environment*. An illustration of a node is given in Figure 1. An application program contains instructions to access shared memory and conditional branch instructions. The mcs processes at all the nodes collectively implement the shared objects that are manipulated by the application programs. The run-time environment at a node executes the shared memory instructions by interacting with the local mcs process; its decision as to which instructions to execute relies on the application program at that node. (We use the term *run-time environment* to refer to the combination of the functionality of a compiler, which sees the whole program, and a conventional run-time system, which makes decisions dynamically based on partial knowledge.) In our framework, consistency conditions are specified as a guarantee on the run-time environments at all nodes with respect to the program at each node. It is the responsibility of both the mcs process and the run-time environment to provide this guarantee.

A straightforward run-time environment would simply submit operations to the mcs one at a time in the order specified by the program. To achieve various optimizations, however, the run-time environment might submit operations out of order, might have multiple operations pending at a time, and might anticipate branches (sometimes incorrectly). We do not address the specific algorithm used by the run-time environment. (That is another very interesting problem, beyond the scope of this paper.) Instead, our goal is to model the run-time environment sufficiently abstractly so that any of a large number of specific run-time environments can fit into this framework. Obviously the run-time environment cannot do just anything—the optimizations that it performs should be transparent to the application program. The

strongest condition we require of a correct run-time environment is that there exist a way (after the fact) to take a subset of the operations performed by the run-time environments at all the nodes and order them to be consistent with some "sequential execution" of the programs at all the nodes. (Some of the operations performed by the run-time environment might end up not being used; for instance, if they resulted from an incorrect prediction about a branch. These operations can be ignored when determining whether there is a corresponding sequential execution.) We emphasize that the order in which operations *appear* to execute is what is important, not the order in which they *actually* execute.

Our framework incorporates rollback and compensating operations in an implicit manner. In particular, we allow the run-time environment to communicate with the mcs in order to perform other operations on the data that are not part of the application programs but that are necessary for achieving the desired consistency condition (for example, operations to restore the state of the shared variables due to incorrect predictions). These operations are ignored when the subset of operations consistent with a sequential execution is taken.

Given this framework, we generalize three known consistency conditions: sequential consistency, weak consistency, and hybrid consistency.

Our second contribution addresses the issue of writing programs for hybrid consistency and formally proving their correctness. *Hybrid consistency* is an efficient and expressive consistency condition [12]; it captures some of the essential features of several other consistency conditions appearing in the literature [3, 15, 18, 19]. Memory access operations are classified as either *strong* or *weak*. This classification is crucial to the following definition of the consistency condition, as observed by the programmer: A global ordering is imposed on strong operations at different processes, but little is guaranteed about the ordering of weak operations at different processes beyond what is implied by their interleaving with the strong operations. The classification also provides hints to the run-time environment concerning which optimizations can be applied to which accesses. Clever use of the classification can improve the performance of programs.

Unfortunately, it is more difficult to program memories that support hybrid consistency than to program memories that support sequential consistency, since the guarantees provided by the former are weaker than those provided by the latter. For instance, how should the programmer decide which accesses should be strong and which should be weak in order to improve performance yet still ensure correctness? A way to cope with this problem is to develop rules and transformations for executing programs that were written for sequentially consistent memories on hybrid consistent memories. The benefit is that sequentially consistent executions are easier to reason about while hybrid consistency can be implemented more efficiently. We consider several techniques for turning programs written for sequential consistency into programs that work for hybrid consistency. Precise, yet short and comprehensible, proofs are provided for the correctness of the programming techniques. Such proofs shed light on the reasons these techniques work; we believe that the insight gained can lead to the development of other techniques.

The first approach we present is based on statically labeling specific accesses as strong according to their type. We prove that programs in which all writes are strong run on hybrid consistent shared memory implementations as if they were sequentially consistent. There is a symmetric result that every hybrid consistent execution of a program in which all reads are strong is sequentially consistent [11]. This theorem requires further assumptions on the execution; although these assumptions are shown

to be necessary, they make the strong read technique impractical.

The second approach is based on the mutual exclusion paradigm and uses the first approach as a tool. One proposed way to program with hybrid-like consistency conditions [18, 21] is to protect accesses to shared data with critical sections and then to use strong operations in the mutual exclusion code and weak operations inside the critical section. When the critical sections are significant in size, the extra cost to execute the strong operations in the mutual exclusion algorithm is more than compensated for by the efficiency of the weak operations in the critical sections. We take a careful look at this paradigm for hybrid consistency and show that it is applicable, although care must be taken. Specifically, we show that many mutual exclusion algorithms designed to work on a sequentially consistent memory can be modified to work correctly on a hybrid consistent memory. The modification is to label all writes in the entry and exit sections as strong and all other operations as weak and to insert a dummy write in an appropriate place. However, this transformation only works for *noncooperative* algorithms in which processes do not participate in the mutual exclusion protocol unless they are actively vying for entry into the critical section.

The third approach for programming with hybrid consistency is to run data-race-free programs. This approach was pioneered by [3, 4, 21, 22, 23] in the context of hardware implementations. (See section 2 for a more detailed discussion.) A *data race* occurs when two accesses to the same location occur, at least one is a write, and there is no synchronization between them. Data races in a program are considered bad practice; they add to the uncertainty of concurrent programs, beyond what is already implied by the fact that different processes may run at different rates and memory accesses may have variable duration. (Some debuggers even regard data races as bugs in the program.) Methods have been developed to detect and report data races, also called *access anomalies*; e.g., [6, 16, 18, 17, 30, 32, 33]. It is reasonable to assume that data-race-free programs account for a substantial portion of all concurrent programs. We formally prove that data-race-free programs run on hybrid consistent shared memory implementations as if they were sequentially consistent; this result is shown in our programmer-oriented framework.

Although many parallel programs are expected to be data-race-free, we cannot ignore the drawbacks of these programs. Proving that a program is data-race-free, even for restricted cases, is co-NP-hard [31]. Also, it is sometimes difficult to find the exact location of the data race in the program [32]. Our static methods provide an alternative and show that it is not necessary to make a program data-race-free in order to guarantee correct behavior. These methods are especially well suited to applications in which reads greatly outnumber writes (or vice versa).

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 describes our system model. The modified definitions of sequential consistency, weak consistency, and hybrid consistency are given in section 4. The static methods for programming with hybrid consistency are discussed in section 5. Data-race-free programs are discussed in section 6. We conclude with a discussion of the results in section 7.

2. Related work. Many existing formal treatments of memory consistency conditions [12, 23, 25, 28, 38] assume that memory operations are executed sequentially—one at a time and in program order. Several recent papers proposed formalisms to allow some nonsequential optimizations [3, 4, 21, 22, 23]. In this section, we compare our work with these formalisms.

Gibbons, Merritt, and Gharachorloo developed a framework for defining consis-

tency condition which is based on a series of I/O automata, and used this framework to define *release consistency*, a formalization of the Stanford DASH shared memory model [22, 23]. This definition, as well as the definition of weak ordering [18, 19], is given at the interface between the mcs and the network. Hence, the resulting consistency conditions are not convenient for programmers and theoreticians. In particular, the definitions in [22, 23] are very detailed and complex. In contrast, in our framework as well as in [12, 25, 28], consistency conditions are defined at the interface between the application program and the system; consistency conditions are defined by describing the way operations are viewed (or ordered) and not the way operations are executed. Hence, our approach is more programmer oriented.

Adve and Hill developed another programmer oriented formalism, called *SCNF* [2, 5], using the assumption that programmers always want to work assuming sequentially consistent memories. In this approach, a consistency condition is a contract between software and hardware. That is, hardware must behave as if it was sequentially consistent for all programs that obey certain properties; various consistency conditions differ in the properties they require from programs. Examples of such conditions include, e.g., DRF0 [3], DRF1 [4], PL_{pc1}, and PL_{pc2} [2]. For each of these programming models, Adve and Hill introduce sufficient conditions for hardware; hardware that follows these conditions executes programs that obey the corresponding programming model as if it was sequentially consistent. The problem with this approach is that the resulting consistency conditions are not defined for programs that do not obey the required properties. This limits the programming style, even in cases where other programming styles could yield better performance. The sufficient conditions introduced by Adve and Hill for DRF0, DRF1, PL_{pc1}, and PL_{pc2} [2] extend the definition of these conditions for all programs. The approach taken by Adve and Hill is to develop a programming model and then suggest how to tailor the hardware to support it. We take the complementary approach, first developing a consistency condition, and then developing programming methods for it.

Our framework includes an explicit model of the program, including control instructions. Our notion of the control flow of the program is derived syntactically from the code. Adve and Hill's sufficient conditions for DRF0, DRF1, PL_{pc1}, and PL_{pc2} [2], which permits nonsequential execution of memory operations, is based on the notion of a read operation controlling a write operation by the same processor. It is not obvious that this notion captures all the possible ways one operation can control another. Other previous work on specifying consistency conditions has either totally ignored control instructions [7, 12, 14, 25, 28] or has only made the informal requirement that uniprocessor control dependencies are preserved [3, 21, 23].

Gibbons and Merritt [22] present a framework that deals formally with pipelining of memory operations. Their framework, recast in our terms, is based on the run-time environment submitting the memory accesses to the mcs, together with a partial order restricting the allowable reorderings of the accesses. Because the interaction between the run-time environment and the mcs is based on partial orders, their framework does not encompass arbitrary out-of-order or speculative execution of operations and it seems difficult to extend it to do so. We have taken a complementary approach and focused on modeling the program and its behavior, leaving unspecified the details of the interaction between the run-time environment and the mcs. Consequently, our framework allows arbitrary out-of-order and speculative execution of operations.

The idea that data-race-free programs can be executed on more relaxed implementations of shared memory as if they were sequentially consistent was pioneered in [2, 3, 4]. Similar results for other consistency conditions were also proved in [10, 22, 23].

We have applied this approach for hybrid consistency, using our framework. Admittedly, hybrid consistency may disallow some optimizations considered in several of the above papers. However, we believe the accessibility of our technical development provides insight concerning the interplay between conditions on the program and memory consistency conditions.

Another approach was taken by Shasha and Snir [37]. They consider multiprocessor programs, written assuming sequential consistency, and investigate where to insert memory barrier operations (fences) so that if operations are executed in pipeline, but the fences are obeyed, then the result is as if the programs were executed sequentially. Their results are based on a partial order that has to hold between instruction instances, representing program order and causality. The execution is sequentially consistent only if this order is acyclic. Therefore, they insert the minimal amount of fences that will make the induced partial order acyclic.

Singh proved sufficient conditions for executions generated by optimized hardware, e.g., pipelined RAM, causal memory, and hybrid consistency, to be sequentially consistent [38]. Unlike the approach taken in this paper and by other researchers in this area, e.g., [2, 3, 4, 10, 22, 23], the approach taken by Singh examines the executions generated by the optimized hardware and not the programs that should be run on it.

The framework developed in this paper was extended in [13] to formally define *alpha consistency*, capturing the semantics provided by DEC-Alpha based multiprocessors (see section 4.1). Two programming methodologies similar to the ones developed in this paper are presented in [13]: (a) every data-race-free program runs on an alpha consistent memory as if it was sequentially consistent, and (b) any noncooperative mutual exclusion algorithm based on sequential consistency can be transformed into a correct solution based on alpha consistency. However, since the Alpha does not support strong operations in the same sense as hybrid consistency (or release consistency),[1] the definition of data-race-free programs and the method for handling mutual exclusion algorithms in [13] are different from those in this paper.

3. Framework. In this section, we present our formal definitions.

3.1. System components. An *application program* consists of a sequence of instructions, each with a unique label. There are two (disjoint) types of instructions: (shared) memory instructions and control instructions. A *memory instruction* specifies an access to a shared object. The specific kind of access depends on the data type of the object.[2] A *control instruction* consists of a *condition* (a boolean function of the process' local state) and a *branch* (jump to the instruction with the given label).

The *memory consistency system* (mcs) implements the shared objects that are manipulated by the application programs. It consists of a process at each node as well as possibly other hardware. Every object is assumed to have a *serial specification* (cf. [25]) defining a set of *(memory) operations*, which are ordered pairs of calls and responses, and a set of *(memory) operation sequences*, which are the allowable sequences of operations on that object. For example, in the case of a read/write object, the ordered pair [$\text{Read}_i(x)$, $\text{Return}_i(x, v)$] forms an operation (p_i reads v from x) for any p_i, x, and v, as does [$\text{Write}_i(x, v)$, $\text{Ack}_i(x)$] (p_i writes v to x). The set of operation sequences consists of all sequences in which every read operation returns

[1] The *memory-barrier* operation of the Alpha does not impose any interprocess ordering of operations.

[2] Our framework does not restrict the data types; however, our programming techniques deal only with read/write operations.

the value of the latest preceding write operation (the usual read/write semantics). The interface to the mcs consists of *calls* (also called invocations) and *responses* on particular objects.

The *run-time environment* at a node takes as input the application program and executes instructions on the mcs. An *operation* is a specific instance of an execution of an instruction. A memory operation consists of two parts: a call (to the mcs process) and a matching response (from the mcs process). A control operation consists of an evaluation of its condition. A control operation is represented by the result (true or false) of the evaluation. Thus the run-time environment must keep track of the local state of the application process in order to perform the evaluation. The run-time environment and mcs process may also communicate concerning issues, such as rollback and compensating operations, necessary to implement certain optimizations. This communication can be modeled with calls to and responses from the mcs process.

An *event* is a call, response, or control operation (condition evaluation). An *execution* (of the system) is a sequence of events such that there is a correspondence between calls and responses (matching object and process), and each response follows its corresponding call.

3.2. Sequential executions. Although an execution is a sequence of events, it can also be viewed as containing operations. Each control operation is itself an event. The memory operations in an execution are obtained by matching up corresponding call and response events; we assume that the run-time environment matches the call and response events defining the memory operations. We also assume that the run-time environment identifies, possibly after the fact, a subset of all the operations executed. This subset consists of the operations that we want to consider as "really happening," and there must be an ordering of this subset that satisfies certain properties. Two important properties, which we present next, are satisfying the serial specifications of the objects (called "legal") and being consistent with a sequential execution of the program (called "admissible").

First, we give a piece of notation. If τ is a sequence of operations and op_i precedes op_j in τ, we write:

$$op_i \xrightarrow{\tau} op_j.$$

A sequence τ of operations is *legal* if for each object x, $\tau|x$, the subsequence of τ consisting of exactly the operations involving x, is in the serial specification of x.

The notion of a sequential execution of the program is formalized with the notion of a flow control sequence for a process p_i. We build up inductively an execution of p_i's program in which every instruction finishes executing before the next one begins. Given process p_i's program, a *flow control sequence*, fcs_i, is a sequence of operations defined as follows. The first element of fcs_i is an instance of the first instruction in p_i's program. Suppose the kth element of fcs_i, denoted by op, is an instance of instruction I in the program. If op is a control operation and its condition evaluates to true, then the $(k+1)$st element of fcs_i is an instance of the instruction whose label is the branch of instruction I. Otherwise the $(k+1)$st element of fcs_i is an instance of the instruction immediately after I in the program. A flow control sequence can be either finite or infinite (for nonterminating programs). Note that the flow control sequence implies a total order on the operations appearing in it; we denote this order by $\stackrel{fcs_i}{\Longrightarrow}$.

Let fcs_i be a flow control sequence for p_i. A sequence τ of memory operations is *fully fcs_i-admissible* if $\tau|i$, the subsequence of τ consisting exactly of operations

involving p_i, is equal to the subsequence of fcs_i consisting exactly of the memory operations. Intuitively, this implies that the ordering of operations by p_i in τ agrees with some flow control sequence fcs_i for p_i and does not end unless the program terminates. A sequence τ of memory operations is *partially fcs_i-admissible* if $\tau|i$ is a prefix of the subsequence of memory operations in fcs_i. Intuitively, this implies that, so far, the ordering implied by the flow control sequence is obeyed by τ, but it is not necessarily completed yet.

A sequence τ of memory operations is *fully* (resp., *partially*) *admissible* with respect to a set of flow control sequences $\{fcs_i\}_{i=1}^{n}$, one for each p_i, if it is fully (resp., partially) fcs_i-admissible for all i.

A sequence of memory operations is a *sequential execution* if it is legal and fully admissible (with respect to some set of flow control sequences).

CLAIM 3.1. *Any legal, partially admissible sequence of memory operations is a prefix of a sequential execution and vice versa.*

3.3. Weak and strong operations. In some consistency conditions it is possible to mark certain instructions in a program as *strong*; all other instructions are *weak*. An instance of a strong instruction is a *strong operation* and an instance of a weak instruction is a *weak operation*. (Strong and weak operations provide different levels of consistency and are part of the definition of hybrid consistency.) In the case of read/write objects this means that it is possible to use strong reads, strong writes, weak reads, and weak writes. In the rest of the paper, we use op_i to denote an operation invoked by p_i (weak or strong), and by sop_i we denote a strong operation invoked by process p_i. We use superscripts, e.g., $op_i^1, op_i^2, \ldots$, to distinguish between operations invoked by the same process (note that the superscript does *not* imply any ordering of the operations). We sometimes use a shorthand notation for read and write operations and denote by $r_i(x, v)$ a read operation (weak or strong) invoked by process p_i returning v from x; we denote by $w_i(x, v)$ a write operation (weak or strong) invoked by process p_i writing v to x. Similarly, $sr_i(x, v)$ is a strong read operation invoked by process p_i returning v from x; $sw_i(x, v)$ is a strong write operation invoked by process p_i writing v to x.

3.4. Control operations and the influence relation. When defining and analyzing consistency conditions, it is important to take into account the effects of control operations. A specific example of the pitfalls associated with failing to do so appears in section 4, where we present our new consistency conditions. To capture the effect of control operations, we define a partial order on operations in a flow control sequence; this partial order is featured in the consistency conditions presented below. Based on the relation fcs_i we define a partial order $\stackrel{co_i}{\Longrightarrow}$ called the *control order*. Formally, for any two memory operations op_i^1 and op_i^2, $op_i^1 \stackrel{co_i}{\Longrightarrow} op_i^2$ if there exists a control operation op_i^3 such that $op_i^1 \stackrel{fcs_i}{\Longrightarrow} op_i^3 \stackrel{fcs_i}{\Longrightarrow} op_i^2$.

We now formalize the notion of one operation influencing another, which relies on the control order. Let τ be a sequence of memory operations and let co_i be a partial order on the operations that is consistent with τ, for each p_i. An operation op_j^1 *directly influences* an operation op_k^2 *in* τ (with respect to the co_i's) if one of the following holds:

1. $op_j^1 \stackrel{co_j}{\Longrightarrow} op_k^2$ and op_j^1 is a read. (Note that $j = k$ in this case.) That is, op_j^1 is a read operation which could affect the execution of op_k^2 through a control operation.
2. $op_k^2 = r_k(y, v), op_j^1 = w_j(y, v)$, $op_j^1 \stackrel{\tau}{\longrightarrow} op_k^2$ and there does not exist $w_h(y, u)$

such that $u \neq v$ and $w_j(y,v) \xrightarrow{\tau} w_h(y,u) \xrightarrow{\tau} r_k(y,v)$. That is, op_k^2 is a read of the value written by op_j^1 and there is no intervening write of a different value.

The *influence* relation is the transitive closure of direct influence. Thus the influence relation is also defined with respect to a set of partial orders. Although we will not usually explicitly mention these partial orders, the influence relation will be used with admissible operation sequences and the relevant partial orders will be the control orders for the corresponding flow control sequences.

Note that an operation directly influences another operation only if it is ordered before it in τ. Since the influence relation is the transitive closure of direct influence, we have the following claim.

CLAIM 3.2. *If op_j^1 influences op_k^2 in τ, then $op_j^1 \xrightarrow{\tau} op_k^2$.*

The following lemma captures the intuition that if a read operation $op_j^1 = r_j(x,v)$ does not influence operation op_k^2, then op_k^2 would have been generated even if op_j^1 had read a value other than v.

LEMMA 3.3. *Let τ be a sequence of memory operations that is partially admissible with respect to a set of flow control sequences $\{fcs_i\}_{i=1}^n$. Let operation op_j^1 in τ be a read $r_j(x,v)$ that does not influence any operation in τ. Let τ' be the result of taking τ and changing op_j^1 to be $r_j(x,w)$ for some $w \neq v$. Then τ' is partially admissible for some set of flow control sequences $\{fcs'_i\}_{i=1}^n$.*

4. Defining nonsequential consistency conditions. In this section, we define three consistency conditions that generalize previously known ones for the nonsequential case. The reason they are generalizations is that in nonoptimized systems, where operations at each process are invoked in program order and only one operation may be pending at a time, fcs_i is simply the sequence of operations in the order they were invoked. We then discuss (by an example) the importance of considering control operations (e.g., if-statements) when specifying consistency conditions.

Sequential consistency [28] is a strong consistency condition stating that there exists a sequential execution that is consistent with the way the actual execution appears to every process. Sequential consistency allows one to reason about a concurrent system using familiar uniprocessor techniques. Since uniprocessor systems are easier to reason about than concurrent systems, this is helpful. Thus many systems (try to) provide sequential consistency. However, providing sequential consistency can incur some costs. For instance, providing sequential consistency in message-based systems requires the response time of some operations to depend on the end-to-end message delay [14, 29]. Later in this paper, we show several programming techniques for achieving sequential consistency when the system only provides a weaker, and preferably cheaper, condition.

DEFINITION 4.1 (sequential consistency). *An execution R is* sequentially consistent *if there exists a subset S of the memory operations in R, a set $\{fcs_i\}_{i=1}^n$ of flow control sequences, and a legal permutation τ of S such that τ is fully admissible with respect to $\{fcs_i\}_{i=1}^n$.*

Note that since τ is a permutation of S, and since τ is fully admissible with respect to $\{fcs_i\}_{i=1}^n$, S must include all operations that appear in all flow control sequences. Hence, S cannot just be chosen arbitrarily. In particular, unless the program is empty, S cannot be empty.

Weak consistency [12, 29] is a very relaxed condition; it does not impose any global ordering on any kind of operations. Its importance lies in the fact that it may be implemented very efficiently, and despite its weakness, there is a large class of

programs for which it suffices. It requires that there exist a subset of the memory operations in the execution and a set of flow control sequences such that for each process, there is a legal permutation of those operations that is consistent with the process' own flow control sequence. Note that each processor may have a different permutation, or "view," of the operations.

DEFINITION 4.2 (weak consistency). *An execution R is* weakly consistent *if there exist a subset S of the memory operations in R and a set of flow control sequences $\{fcs_i\}_{i=1}^{n}$, one for each p_i, such that for each p_i, there exists a legal permutation τ_i of S that is fully fcs_i-admissible.*

Hybrid consistency [12] is intermediate between sequential and weak consistency; it combines the expressiveness of the former and the efficiency of the latter. Hybrid consistency distinguishes between two types of operations—strong and weak. It states that there must be a subset of the memory operations in the execution, a total order on the strong operations among them, and a set of flow control sequences satisfying the following. For each process, there is a legal permutation of the operations in the subset that is consistent with four orders: the process' own flow control sequence, every other process' control order, the total order on the strong operations, and the relative order of every pair of strong and weak operations by another process in that process' flow control sequence. Furthermore, all accesses of the same process to the same location will be viewed by all the processes in the same order. It is possible to implement hybrid consistency in such a way that weak operations are extremely fast [12].

DEFINITION 4.3 (hybrid consistency). *An execution R is* hybrid consistent *if there exist a subset S of the memory operations in R, a set of flow control sequences $\{fcs_i\}_{i=1}^{n}$, and a permutation ρ of the strong operations in S such that for each process p_i, there exists a legal permutation τ_i of S with the following properties:*

1. *τ_i is fully fcs_i-admissible, i.e., it is consistent with the process' own flow control sequence.*
2. *If $op_j^1 \overset{co_j}{\Longrightarrow} op_j^2$, then $op_j^1 \overset{\tau_i}{\longrightarrow} op_j^2$, for any j, i.e., it is consistent with every other process' control order.*[3]
3. *If $op_j^1 \overset{fcs_j}{\Longrightarrow} op_j^2$ and at least one is strong, then $op_j^1 \overset{\tau_i}{\longrightarrow} op_j^2$, for any j, i.e., it is consistent with the relative order of every pair of strong and weak operations by another process in that process' flow control sequence.*
4. *If $op_j^1 \overset{\rho}{\longrightarrow} op_k^2$ (implying both are strong), then $op_j^1 \overset{\tau_i}{\longrightarrow} op_k^2$, for any j and k, i.e., it is consistent with the total order on the strong operations.*
5. *If $op_j^1 \overset{fcs_j}{\Longrightarrow} op_j^2$ and op_j^1 and op_j^2 access the same location, then $op_j^1 \overset{\tau_i}{\longrightarrow} op_j^2$, for any j, i.e., all accesses of the same process to the same location will be viewed by all the processes in the same order.*

We now discuss the last property in more detail. It states that all operations on the same object by the same process p_j are viewed by every other process in the same order as they are viewed by p_j. This property does not appear in the original definition of hybrid consistency [12]; however, it is necessary in order for some of our results to hold, as is shown in section 6.3. Evidence suggests it is a reasonable assumption, since some previous authors make the even stronger assumption that all processes view all operations on the same object, no matter which process invoked them, in the same order.[4]

[3] This condition is not part of the original definition of hybrid consistency [12], which did not include control operations.

[4] In [3, 22, 23], a total order on all the writes to the same location is assumed. In addition, it is

To illustrate the problem that occurs if hybrid consistency were defined without considering control operations, consider the following example, assuming x and y are initially 0 and all instructions are weak:

p_1's program	p_2's program
$\text{tmp}_1 := \text{read}(x)$;	$\text{tmp}_2 := \text{read}(y)$;
if $\text{tmp}_1 = 5$ then	if $\text{tmp}_2 = 5$ then
$\text{write}(y, 5)$;	$\text{write}(x, 5)$;

In this program, the values returned by the reads affect the decision of whether to invoke the writes, and the invocations of the writes affect the possible values returned by the reads. For example, assume this program is being executed on a hybrid consistent memory and all operations are weak. If we ignore the existence of control instructions, the definition would allow the following execution R:

$$R|p_1 = r_1^1(x,5), w_1^2(y,5),$$
$$R|p_2 = r_2^1(y,5), w_2^2(x,5).$$

To show R is hybrid consistent we take

$$\tau_1 = w_2^2(x,5), r_1^1(x,5), w_1^2(y,5), r_2^1(y,5),$$
$$\tau_2 = w_1^2(y,5), r_2^1(y,5), w_2^2(x,5), r_1^1(x,5).$$

Note that condition 2 is not satisfied, since the sequences do not preserve the order of operations separated by control operations.

The program for p_1 allows the read(x) to return 5, whereas x could have taken the value 5 only if p_1 writes 5 to y, which will happen only if p_1 reads 5 in x. It is important to eliminate such executions with circular inference relations, where the prediction about the result of a control operation could affect its actual result.[5] If such behavior is allowed, then writing programs and arguing about them becomes almost impossible.

One might want to allow the programmer to choose whether or not a control operation should enforce an ordering. This can be done in a manner similar to the distinction we make between strong and weak memory operations. Introducing "weak" control operations in this manner further complicates the programming model; therefore, we have decided not to include them in our model.

4.1. Discussion of the definitions.

Viewing order vs. execution order. We would like to emphasize that the order in which operations may be viewed by different processes need not reflect the order in which they are executed. For example, consider the program in Figure 2. Due to the control order, every process must view the write to z after the read from x. On the other hand, a sophisticated run-time environment (or compiler) could detect that the write to z would be invoked regardless of the outcome of the read from x. Therefore,

---

assumed that a value read from a specific location can be uniquely identified with a write operation. Thus, all the processes view all the writes to the same location in the same order. Since a value read from a specific location can be uniquely identified with a write operation, each read is viewed by all the processes to be between the same writes to that location. These two assumptions imply that all the processes view all the operations on the same object in the same order.

[5] See [37] for more discussion of circular relations in the execution of programs.

```
p1's program
tmp := read(x);
if tmp = v then
     write(y, u);
write(z, w);
```

FIG. 2. *Viewing order vs. execution order.*

p_1's program	p_2's program
*tmp$_1$:= read(&x);	*tmp$_2$:= read(&y);

FIG. 3. *Limitations of the framework.*

the run-time environment can invoke the write to z before the read from x. This is allowed by the definition of hybrid consistency since it is possible to order the write to z after the read from x (even if they were executed in reverse order).

Limitations of the framework. Our framework for defining consistency conditions does not support dependencies through registers and indirect addressing. For example, the program in Figure 3 is not handled by our framework. (We use & and * for indirect addressing in the style of C/C++.) Followup work [13] extends the framework to include RISC-type addressing, i.e., registers can be used to provide the data or the address for an operation (instead of just constants, as in this paper). It shows that all the results of this paper hold in the extended framework as well, although the definitions are slightly more complicated.

Programmability vs. hardware optimizations. The definition of hybrid consistency includes several choices which reflect our opinions about the guarantees required by programmers from a consistency condition. These choices may disallow some hardware optimizations. We would like to stress, however, that these choices are not an inherent part of the framework. Specifically, if a particular choice we make does not allow an invaluable hardware optimization, then the definition of the consistency condition can be easily changed to support this optimization.

As an example, consider the program in Figure 4. In this program, the two writes by p_1 are labeled as strong, while the two reads of p_2 are labeled as weak; assume that the initial values of x and y are 0. The definition of hybrid consistency requires that every process views its own weak reads in the same order as they appear in its flow control sequence. Hence, if the first read by p_2 returns 1 from y, then the second read by p_2 cannot return 0 from x.

We believe that this is the semantics expected by programmers. However, this implies that an implementation of hybrid consistency cannot afford to execute weak reads out of order unless it provides rollback. This happens due to the fact that when the run-time system decides on the execution order for two weak reads by the same process, it may not know whether there are concurrent strong writes to the same objects (as in Figure 4). Yet, if it important to allow hardware optimizations that require a process to view its own weak reads out of order; then the definition of hybrid consistency can be easily changed by removing the first condition in Definition 4.3. We remark that in this case, the result about running data-race-free programs (Theorem 6.6) remains correct, but the programming techniques developed in section 5 are

p_1's program	p_2's program
swrite$(x, 1)$;	read(y);
swrite$(y, 1)$;	read(x);

FIG. 4. *The ordering of weak reads.*

no longer valid.

5. Static approach. In this section, we present techniques for writing programs for hybrid consistent shared memories that are based on statically classifying accesses according to their type (read or write) and the object they access. In section 5.1, we show that every hybrid consistent execution of a program in which all writes are strong is sequentially consistent. This result is used to develop an efficient synchronization code in section 5.2. In [11], we present the symmetric result that every hybrid consistent execution of a program in which all reads are strong is sequentially consistent. In order to prove this theorem, further assumptions on the execution must be made. Although these assumptions are shown to be necessary, they make the result impractical.

5.1. Strong writes. We prove the following theorem.

THEOREM 5.1. *Every hybrid consistent execution of a program in which all writes are strong and all reads are weak is sequentially consistent.*

Proof. Let R be such an execution. Let S be a subset of memory operations in R, fcs_i for each p_i be a flow control sequence, ρ be a permutation of the strong operations (namely, the writes) in S, and τ_i for each p_i be a legal permutation of S as guaranteed by the definition of hybrid consistency. We will insert the (weak) read operations into ρ to construct τ, a legal permutation of S that is fully admissible with respect to the fcs_i's.

A read by process p_i is inserted after any write that precedes it in τ_i and before any write that follows it in τ_i. This can be done since τ_i agrees with ρ on the order of strong operations. Furthermore, it will be inserted after any read by p_i that precedes it in τ_i and before any read by p_i that follows it in τ_i. The ordering of read operations by different processes is unimportant.

Clearly, τ is a permutation of S. Since τ_i preserves the ordering of operations by p_i, it follows that τ is fcs_i-admissible for all i.

The fact that τ is legal follows from the fact that τ_i is legal, that τ_i agrees with ρ on the order of strong operations, and from the construction of τ. □

5.2. Using strong writes to program with critical sections. Theorem 5.1 is useful for designing and proving correctness of programs which rely on hybrid consistency. A simple way to use it is to take a program which is designed for sequential consistency, label each write as a strong write and each read as a weak read, and run it on a hybrid consistent memory. However, there is even a more efficient way to employ the above theorem. If the program has explicit synchronization code dedicated to coordinating memory access operations, while the rest of the code ignores synchronization issues, then it is possible to apply Theorem 5.1 only to the synchronization code and label all other memory accesses—both reads and writes—as weak. We demonstrate this method for mutual exclusion. Given a mutual exclusion algorithm designed for sequentially consistent memories, we produce a modified algorithm by adding one strong write (to a location which is never read from) to the entry section and by labeling all the writes in the synchronization part of the code as strong, while

all other memory accesses are labeled as weak. We prove that the modified algorithm guarantees mutual exclusion (in a strong sense) on hybrid consistent memories.

Several papers on relaxed consistency conditions refer to the common method of programming with critical sections as a justification for the separation of strong and weak operations. In [21, p. 19], it is argued that

> for example, a large class of programs are written such that accesses to shared data are protected within critical sections. Such programs are called *synchronized programs*, whereby writes to shared locations are done in a mutually exclusive manner (no other reads or writes can occur simultaneously). In a synchronized program, all accesses (except accesses that are part of the synchronization constructs) can be labeled as *ordinary*$_L$.

Our result shows that there is a more efficient way to utilize synchronized programs. If the mutual exclusion algorithm is noncooperative, then it is not necessary to label reads that are part of the synchronization constructs (i.e., entry and exit sections) as strong (provided that an extra write is added).

We remind the reader that an algorithm for mutual exclusion consists of four disjoint sections for each process—entry, critical, exit, and remainder (cf. [36]). In the *entry* section, a process tries to gain access to the *critical* section; the *exit* section is executed by each process upon leaving the critical section; the *remainder* section is the rest of the code. Processes cycle through the four sections of their code. Informally, an algorithm guarantees mutual exclusion provided that the following hold:

Mutual exclusion: no two processes are inside the critical section at the same time, and

Deadlock freedom: if there exists a process that is continually in its entry section from some point on, then there exists another process that enters (and leaves) its critical section infinitely often.

A mutual exclusion algorithm provides code for the entry and exit sections. It should treat the code in the critical and remainder sections as black boxes (with some restrictions, as discussed below).

The above definition of mutual exclusion assumes that the order in which operations are executed reflects the order in which they are viewed by the processes. However, our formalism for defining consistency conditions puts restrictions only on the order in which operations are viewed, which may not correspond directly to the order in which they are executed. To be able to cope with these conditions, we define *logical mutual exclusion* as follows. Given a mutual exclusion algorithm (program), consider a flow control sequence for process p_i, and let CS_i^k be the set of operations invoked by process p_i during the kth time that p_i executes the critical section in this sequence.

Algorithm $\mathcal{A}$ guarantees *logical mutual exclusion based on a consistency condition* C provided that the following hold. Let σ be an execution of $\mathcal{A}$ that is allowed by C and let $\{\tau_r\}_{r=1}^n$ be a set of sequences as required in the definition of C (the views of the different processes). Consider the CS_i^k's induced by $\overset{fcs_i}{\Longrightarrow}$.

Logical mutual exclusion: For any four operations $op_i^1, op_i^2 \in CS_i^k$ and $op_j^1, op_j^2 \in CS_j^l$, $op_i^1 \overset{\tau_q}{\longrightarrow} op_j^1$ if and only if $op_i^2 \overset{\tau_s}{\longrightarrow} op_j^2$, for all processes p_q and p_s. As before, this implies that there is a total order on all critical section executions; furthermore, this ordering is the same in all τ_r's.

Deadlock freedom: For every process p_q, if τ_q is infinite and there exists a process p_i that is in its entry section from some point on in τ_q, then there is another

process p_j that enters (and leaves) its critical section infinitely often in τ_q.

In the rest of this section we mean logical mutual exclusion whenever we refer to mutual exclusion.

Our result assumes that the mutual exclusion algorithm does not communicate with the algorithm that is executed inside the critical section or remainder section. Specifically, we assume that variables that are accessed in the entry or exit section are not accessed in the critical or remainder sections. To capture this property, let the *exclusion set* of a mutual exclusion algorithm $\mathcal{A}$ be the set of shared variables read in the entry or exit sections of $\mathcal{A}$; this set is denoted $exc(\mathcal{A})$.

DEFINITION 5.2. *A mutual exclusion algorithm $\mathcal{A}$ is* noncooperative *if every process which executes the critical section or the remainder section of $\mathcal{A}$ does not write any variable in $exc(\mathcal{A})$; otherwise, the algorithm is* cooperative.[6]

We assume that the mutual exclusion algorithm is designed to run with any code in the critical section (subject to the above restriction) and that it can be run in asynchronous systems (i.e., the algorithm does not depend on any timing behavior).

CLAIM 5.3. *There is at least one write to a variable in $exc(\mathcal{A})$ in the exit section of every* noncooperative *mutual exclusion algorithm based on sequential consistency.*

Proof. Assume, by way of contradiction, that there exists a noncooperative mutual exclusion algorithm $\mathcal{A}$ such that there is no write to any of the variables of $exc(\mathcal{A})$ in the exit section of $\mathcal{A}$. Assume that we run the algorithm with the following critical section (for every process p_i), under the assumption that x is initially 0:

$tmp_i = r(x)$,
$tmp_i := tmp_i + 1$,
$w(x, tmp_i)$.

Recall that tmp_i and x are not accessed in the entry and exit sections of $\mathcal{A}$. Consider now a sequentially consistent execution R of $\mathcal{A}$ with this critical section in which p_0 starts at time 0 and runs until it enters the critical section, completes it, and exits; then p_1 starts and runs until it enters the critical section, completes it, and exits. Denote by t the time at which p_0 completes the read operation from x.

We claim that there is a sequentially consistent execution R' of $\mathcal{A}$ with this critical section in which p_0 starts at time 0 and runs until time t, when it completes the read operation; then p_1 starts and runs until it enters the critical section, completes it, and exits; then p_0 completes its critical section and exits. R' exists because we assumed that the algorithm is asynchronous. Note that there is no write in the exit section of $\mathcal{A}$ and there is no write to variables in $exc(\mathcal{A})$ in the critical section of p_0. Thus, all the values read by p_1 in its entry section in R' are equal to the values it reads in its entry section in R; since, in R, p_1 must eventually enter the critical section (to avoid deadlock), it must eventually enter the critical section in R'.

Clearly, R' violates the definition of mutual exclusion, since p_0 and p_1 are inside the critical section at the same time. Furthermore, note that in R' both p_0 and p_1 read the value 0 from x and thus the final value of x at the end of R' is 1. This final value could not be obtained in any execution that preserves logical mutual exclusion. □

Given a noncooperative algorithm $\mathcal{A}$ for the mutual exclusion problem based on sequential consistency, label every write in the entry and exit sections as strong and

[6] The definition of noncooperative algorithms given in [12] allows processes that are executing the critical section to access variables in $exc(\mathcal{A})$. The additional requirement made here seems quite reasonable, since we usually want to treat the mutual exclusion algorithm as a general solution, independent of the rest of the code, and to use it as a subroutine. This is also the approach taken in many of the known solutions to the mutual exclusion problem [36]. A more detailed discussion and motivation for noncooperative mutual exclusion algorithms appears in [12].

every read in the entry and exit sections as weak; operations inside the critical section or remainder section are labeled as weak. Next, add a strong write to some object $nac(\mathcal{A})$ that is not accessed elsewhere in the program such that this write will be the last instruction executed (in a flow control sequence) before each critical section. Call this modified algorithm $\mathcal{A}'$.

In order to prove that $\mathcal{A}'$ guarantees mutual exclusion based on hybrid consistency, we have to show that it is free of deadlocks and that it guarantees logical mutual exclusion.

LEMMA 5.4. *$\mathcal{A}'$ guarantees deadlock freedom based on hybrid consistency.*

Proof. Assume, by way of contradiction, that there exists a hybrid consistent execution σ of $\mathcal{A}'$ that has a deadlock. Denote by $\{\tau_j\}_{j=1}^n$ the set of sequences of operations as guaranteed by the definition of hybrid consistency. Let σ' be the execution that results by eliminating from σ all operations that are not invoked inside (w.r.t. the flow control sequences) the entry or exit section of $\mathcal{A}'$. Since the algorithm is noncooperative, there are no writes to variables in $exc(\mathcal{A}')$ outside (w.r.t. the flow control sequences) the entry and exit sections of $\mathcal{A}'$. Thus, σ' includes all the writes (in σ) to variables in $exc(\mathcal{A}')$, and thus, σ' is hybrid consistent. Specifically, define for each j a flow control sequence fcs'_j by eliminating from fcs_j all the operations of p_j that do not appear in σ'. Thus, the set of sequences $\{\tau'_j\}_{j=1}^n$ that results from eliminating all the operations that do not appear in σ' from $\{\tau_j\}_{j=1}^n$ obeys all the requirements in the definition of hybrid consistency (w.r.t. the $\{fcs'_j\}_{j=1}^n$). Note that all the writes in σ' are strong, and thus, by Theorem 5.1, σ' is sequentially consistent. Furthermore, in σ' there is a deadlock. Since there are no reads from $nac(\mathcal{A})$, the execution σ'' that results from eliminating all the writes to $nac(\mathcal{A})$ is also sequentially consistent and there is a deadlock in σ''. Note that σ'' is an execution of $\mathcal{A}$ in the case of an empty critical section. Thus, there is a sequentially consistent execution of $\mathcal{A}$ that has a deadlock. This is a contradiction. □

LEMMA 5.5. *$\mathcal{A}'$ guarantees logical mutual exclusion based on hybrid consistency.*

Proof. Recall that a sequential execution is a sequence of operations. Hence, each sequential execution of a program can be viewed by itself as a sequence τ of operations that obeys the requirements in the definition of sequential consistency. Therefore, since $\mathcal{A}$ guarantees mutual exclusion based on sequential consistency, every sequential execution of $\mathcal{A}$ must guarantee logical mutual exclusion as defined above. We will show that if $\mathcal{A}'$ does not guarantee mutual exclusion based on hybrid consistency, we may build a sequential execution of $\mathcal{A}$ in which mutual exclusion is violated.

Let σ be a hybrid consistent execution of an instance of $\mathcal{A}'$ in which every execution of the critical section consists of two writes, and let $\{\tau_j\}_{j=1}^n$ be the set of sequences of operations as guaranteed in the definition of hybrid consistency.

Call the last operation executed by a process before entering the critical section the *entry point*. Note that the entry point is a write operation on $nac(\mathcal{A})$. By Claim 5.3, there is at least one write in the exit section. Call the first write in the exit section the *exit point*. Each pair of matching entry and exit points is called a *critical pair*. We now show that in each τ_i all the critical pairs are ordered in a nonoverlapping way, which gives the result for critical section executions.

Assume, by way of contradiction, that there exists a sequence τ_j in which two critical pairs overlap. Assume that $[op_k^1, op_k^2]$ and $[op_l^1, op_l^2]$ are critical pairs that overlap, and assume, without loss of generality, that $op_k^2 \xrightarrow{\tau_j} op_l^2$. Let σ' be the execution that results by eliminating from σ all operations that are not invoked inside entry and exit sections of $\mathcal{A}'$. Since the algorithm is noncooperative, σ' includes all the writes (in σ) to variables in $exc(\mathcal{A}')$, and thus, σ' is hybrid consistent. By Theorem

5.1, σ' is sequentially consistent. Thus, we may build a legal sequence of operations τ in the same way as in the proof of Theorem 5.1. That is, the strong writes are ordered in τ in the order they appear in any sequence τ_i. A read by process p_i is inserted after any write that precedes it in τ_i and before any write that follows it in τ_i. Furthermore, it will be inserted after any read by p_i that precedes it in τ_i and before any read by p_i that follows it in τ_i. The ordering of read operations by different processes in unimportant. Note that τ is a sequential execution of $\mathcal{A}'$ in the case of an empty critical section.

By the construction of τ, and since we assumed that $[op_k^1, op_k^2]$ and $[op_l^1, op_l^2]$ are overlapping in τ_j, the latest of op_k^1 and op_l^1 precedes both op_k^2 and op_l^2 in τ. Consider the prefix τ' of τ that ends with the latest of op_k^1 and op_l^1. Note that any operation in τ' that is invoked during the exit sections that correspond to $[op_k^1, op_k^2]$ and $[op_l^1, op_l^2]$ is a read. Remember that there are no reads from $nac(\mathcal{A})$. Thus, τ'', the result of eliminating from τ' all reads that are invoked during the exit sections that corresponds to $[op_k^1, op_k^2]$ and $[op_l^1, op_l^2]$ and all writes to $nac(\mathcal{A})$, is a prefix of a sequential execution of $\mathcal{A}$. Denote the writes in the critical section that corresponds to $[op_k^1, op_k^2]$ by w_k^1 and w_k^2 and the writes in the critical section that corresponds to $[op_l^1, op_l^2]$ by w_l^1 and w_l^2. Add w_k^1, w_l^1, w_k^2, and w_l^2 in this order to τ'' to form τ'''. Since all operations that were added to τ'' are writes, τ''' is legal and is therefore a prefix of a sequential execution. Moreover, $w_k^1 \xrightarrow{\tau'''} w_l^1$ but $w_l^1 \xrightarrow{\tau'''} w_k^2$, which is a violation of logical mutual exclusion. Thus, τ''' can be extended to a sequential execution of $\mathcal{A}$ that violates mutual exclusion. This is a contradiction to the assumption that every sequential execution of $\mathcal{A}$ guarantees logical mutual exclusion.

Thus, we have shown that all critical pairs appear in every sequence τ_j in a nonoverlapping manner. Since the exit points are strong writes, all the processes agree (in the τ_j's) on the same nonoverlapping order for the critical pairs. Hence, we need only to show that the order in which critical pairs are ordered in every sequence τ_j is unique.

Let cp, cp', and cp'' be three critical pairs such that cp is ordered before cp' and after cp'' in any τ_i. Since the exit point of cp is a strong operation, and since it is ordered before the entry point of cp', all the operations of cp are ordered before the entry point of cp'. Since the entry point of cp is a strong operation, and since it is ordered after the exit point of cp'', all the operations of cp are ordered after cp''. Thus, (logical) mutual exclusion is also guaranteed by the algorithm. □

Note that, in cooperative algorithms, a process that participates in the mutual exclusion protocol, but does not wish to enter the critical section, can access variables in the exclusion set while executing the remainder section. If these accesses are labeled as weak, the solution might not be correct anymore. A simple example (of a token passing mutual exclusion algorithm) is detailed in [11].

6. Running data-race-free programs. In this section we prove that data-race-free programs behave on hybrid consistent memory implementations as if they were sequentially consistent. Hybrid consistency is a weaker condition than sequential consistency, and can be implemented more efficiently [12, 14, 29]. Clearly, having data-race-free programs behave on hybrid consistent memory implementations as if they were sequentially consistent is a desirable property, since many concurrent programs attempt to be data-race-free. In our opinion, the proofs in this section are more transparent than proofs of similar results using other formalisms [22, 23]. This also demonstrates the usefulness of our framework for investigating and proving properties of consistency conditions in general (cf. [13]).

6.1. Definition of data-race-free programs. Let op_i^1 and op_j^2 be two operations appearing in some sequence of memory operations α. Then

- $op_i^1 \overset{po}{\Longrightarrow} op_j^2$ if $i = j$ and $op_i^1 \overset{\alpha}{\longrightarrow} op_j^2$.
- $op_i^1 \overset{so}{\Longrightarrow} op_j^2$ if both op_i^1 and op_j^2 are strong operations and $op_i^1 \overset{\alpha}{\longrightarrow} op_j^2$.

The relation *happens before*, denoted by $\overset{hb}{\Longrightarrow}$, is the transitive closure of the union of $\overset{po}{\Longrightarrow}$ and $\overset{so}{\Longrightarrow}$. This definition is similar to the definition of *happens before* in [3] and is closely related to the definition of *happened before* defined by Lamport [27] for message passing systems.

Two memory accesses *conflict* if they both access the same memory location and at least one of them is a write. A *data race* occurs in a sequence of memory operations when two conflicting memory accesses are not ordered by the happens before relation.

DEFINITION 6.1. *A program is* data-race-free *if none of its sequential executions contains a data race.*

These definitions are modeled after those in [3].

6.2. Running data-race-free programs on hybrid consistent memory. We prove that every hybrid consistent execution of a data-race-free program is sequentially consistent.

To prove this result, we consider a legal sequence of memory operations τ_i, as guaranteed for some process p_i in the definition of hybrid consistency, that is *minimal* with respect to the number of *switched* operations (operations by the same process p_j whose order in τ_i is not consistent with p_j's flow control sequence). We show that if τ_i is not fully admissible (i.e., a sequential execution), then there exists a prefix of a sequential execution of the program that contains a data race. If τ_i is not fully admissible, it must contain at least one pair of switched operations. We locate the "first" pair of switched operations in τ_i, such that no other pair of switched operations is ordered between them. Because τ_i is minimal we know this pair was switched to preserve legality. This fact is used to show that there is a data race between some pair of operations that precedes this switched pair. Our main problem is to place these two operations (and the data race between them) in a legal and partially admissible sequence. This is done by taking the two operations and the operations that influence them and ordering them as in τ_i, and adding any operations necessary to preserve the flow control sequences of all processes. The key point to prove about the resulting sequence is its legality. In doing so, we either change the value that a read returns (and invoke Lemma 3.3), or, if this does not help, we show that there is a data race earlier in the sequence. Thus we have constructed a prefix of a sequential execution with a data race, which is a contradiction. The details follow.

Fix a data-race-free program `Prog`, a hybrid consistent execution R of `Prog`, and as guaranteed by the definition of hybrid consistency, a subset S of the memory operations and a flow control sequence fcs_i for each process p_i.

Fix an arbitrary process p_i. Let T be the set of all operation sequences τ_i that satisfy Definition 4.3 (hybrid consistency). For τ_i in T, an ordered pair of operations of p_j, $\langle op_j^2, op_j^1 \rangle$, is *switched* in τ_i if $op_j^2 \overset{\tau_i}{\longrightarrow} op_j^1$, but $op_j^1 \overset{fcs_j}{\Longrightarrow} op_j^2$.

Let τ_i be some element of T with a minimal set of switches. That is, there does not exist $\tau_i' \in T$, such that the set of pairs of switched operations of all processes in τ_i' is strictly contained in the set of pairs of switched operations of all processes in τ_i.

We prove the main theorem of this section by way of contradiction, using the following lemma.

LEMMA 6.2. *If τ_i is not fully admissible, then there exists a prefix of a sequential*

execution of `Prog` *which contains a data race.*

Proof. Assume that τ_i is not fully admissible. We first prove the following claim, which locates the pair of operations that is a candidate for a data race.

CLAIM 6.3. *There exist two operations* $op_j^2(x,v)$ *and* $op_k^1(x,w)$ *in* τ_i *such that*

1. $op_j^2(x,v) \xrightarrow{\tau_i} op_k^1(x,w)$,
2. *there is a data race between* $op_j^2(x,v)$ *and* $op_k^1(x,w)$ *in* τ_i, *and*
3. *there is no pair of switched operations in* τ_i *up to* $op_k^1(x,w)$.

Proof. Since τ_i is not fully admissible, there exists at least one pair of switched operations in τ_i. Let $\langle op_j^2(x,v), op_j^1(y,u)\rangle$ be the first ordered pair of switched operations, i.e., there is no other pair of switched operations which is completely ordered before $op_j^1(y,u)$ in τ_i. If there is more than one ordered pair that share the same second operation (namely, op_j^1), then choose the pair whose first operation is latest.

Since τ_i is minimal, $op_j^2(x,v)$ and $op_j^1(y,u)$ are switched in order to preserve legality.

Since the operations are by the same process p_j, and since the order of operations by the same process to the same object is preserved (last property in Definition 4.3), it follows that $x \neq y$. Thus, there exists an operation $op_k^1(x,w)$ which conflicts with $op_j^2(x,v)$ and $op_j^2(x,v) \xrightarrow{\tau_i} op_k^1(x,w) \xrightarrow{\tau_i} op_j^1(y,u)$.

If there is a strong operation by p_j between op_j^2 and op_j^1 (including one of them), then the third property in Definition 4.3 implies that they cannot be switched. Therefore, there are no strong operations by p_j between op_j^2 and op_j^1 in τ_i, including op_j^2 and op_j^1.

It is also the case that $j \neq k$. Suppose otherwise. Either op_j^2 and op_k^1 are switched or else op_k^1 and op_j^1 are switched. In the first case, $\langle op_j^2, op_k^1\rangle$ would have been the pair chosen and in the second case, $\langle op_k^1, op_j^1\rangle$ would have been the pair chosen.

Since $j \neq k$, op_j^2 and op_k^1 conflict, and there is no strong operation by p_j between them, there is a data race between op_j^2 and op_k^1 in τ_i. Thus the claim is shown. □

Let I be the set of operations in τ_i that influence either op_j^2 or op_k^1 (including op_j^2 and op_k^1 themselves). Let τ_i' be the shortest prefix of τ_i that includes every operation in I. Let π be the subsequence of τ_i' which contains exactly the set of all operations in I. The following claim shows that π is consistent with $\overset{fcs_l}{\Longrightarrow}$ for all l.

CLAIM 6.4. *For every pair of operations* op_l^1 *and* op_l^2, *if* $op_l^1 \xrightarrow{\pi} op_l^2$, *then* $op_l^1 \overset{fcs_l}{\Longrightarrow} op_l^2$.

Proof. By Claim 3.2, the influence relation is consistent with $\xrightarrow{\tau_i}$. Therefore, no operation in I follows op_k^1 in τ_i, implying that op_k^1 is the last operation in τ_i'. By condition 3 in Claim 6.3, τ_i' does not include any pair of switched operations, and neither does π. □

Now, we construct a new partially admissible (with respect to the fcs_i's) sequence π' by adding to π every operation op_l^1 not in π, such that $op_l^1 \overset{fcs_l}{\Longrightarrow} op_l^2$ and op_l^2 is in π, for each process l, as follows. For every such operation that was originally in τ_i', add it in the same place as in τ_i'. For all other operations, add them arbitrarily, maintaining consistency with $\overset{fcs_l}{\Longrightarrow}$, for all l. By the definition of hybrid consistency, every added operation op_l^1 that is not in τ_i' must be weak. This is because op_l^1 precedes op_l^2 in $\overset{fcs_l}{\Longrightarrow}$, but follows op_l^2 in τ_i, a possibility precluded of any strong operation by hybrid consistency (condition 3 in Definition 4.3).

We would now like to determine if π' is a legal sequence. By the definition of the

influence relation and τ_i, π is a legal sequence. In particular, all reads in π are legal. Thus, if there are illegal reads in π', then each illegal read is either a read that was added to π, or a read that became illegal because of some write that was added to π.

Consider every illegal read r_l in π' that was added to π. We claim that r_l does not influence any operation in π', with respect to the co_i's. Suppose it does. Since r_l is a read, it must influence an operation through a control relation, i.e., $r_l \stackrel{co_l}{\Longrightarrow} op_l$, for some l. Since r_l is not in I, op_l is not in I. Thus op_l was included in π' because there exists op'_l in I such that $op_l \stackrel{fcs_l}{\Longrightarrow} op'_l$. This implies that $r_l \stackrel{co_l}{\Longrightarrow} op'_l$ and that r is in I. We therefore have a contradiction.

Now, for each of the illegal reads in π' that were added to π, change its value to match the value of the most recent write to the same object. (Note that this can be done since an illegal read in π' does not influence any other operation in π'.) Call this sequence π''. Apply Lemma 3.3 successively for each fixed-up read, starting with π', to deduce that π'' is partially admissible with respect to some set of flow control sequences $\{fcs'_l\}_{l=1}^{n}$. So, if there are no illegal reads in π'', then π'' is legal and it follows that it is a prefix of a sequential execution of `Prog`.

To complete the proof of the lemma where π'' is legal, we now show that π'' contains a data race between $op_j^2(x, v)$ and $op_k^1(x, w)$. By Claim 6.3, there is no strong operation by p_j between op_j^2 and op_k^1 inclusive in τ'_i; also, note that no strong operation was added to π. Also, $j \neq k$; thus op_j^2 and op_k^1 are not ordered by the happens-before relation.

We are left with the case in which there is an illegal read in π''. It became illegal due to the insertion of some write. Let $r_q^1(z, v_1)$ be the first illegal read in π'', and let $w_p^1(z, v_2)$, $v_1 \neq v_2$ be the corresponding inserted write. Denote by σ the shortest prefix of π'' which includes $r_q^1(z, v_1)$. Let σ' be the same sequence where $r_q^1(z, v_1)$ is replaced by $r_q^2(z, v_2)$. We complete the proof by showing the following claim.

CLAIM 6.5. *σ' is a prefix of a sequential execution of* `Prog` *which contains a data race between r_q^2 and w_p^1.*

Proof. We first show that $r_q^1 \stackrel{\tau_i}{\longrightarrow} w_p^1$. Assume otherwise that $w_p^1 \stackrel{\tau_i}{\longrightarrow} r_q^1$. Since w_p^1 is not in I, it must not influence r_q^1 in τ_i. Therefore, there is another write $w_r^2(z, v_1)$ ordered between w_p^1 and r_q^1 which influences r_q^1 in τ_i. But then w_p^1, w_r^2, and r_q^1 would be ordered similarly in π'. This gives a contradiction to the assumption that r_q^1 becomes illegal due to the insertion of w_p^1.

So we can assume that $r_q^1 \stackrel{\tau_i}{\longrightarrow} w_p^1$; it follows that w_p^1 is not in τ'_i, implying that w_p^1 is weak. We claim that $p \neq q$. Otherwise, if $p = q$, then $w_p^1(z, v_2) \stackrel{fcs_p}{\Longrightarrow} r_q^1(z, v_1)$. By the last property of hybrid consistency, it follows that $w_p^1(z, v_2) \stackrel{\tau_i}{\longrightarrow} r_q^1(z, v_1)$, which is a contradiction. Therefore $p \neq q$.

We now show that $r_q^1(z, v_1)$ and $w_p^1(z, v_2)$ are not ordered by the happens before relation in π'', and hence, they are not ordered by the happens before relation in σ. Suppose they were. Then, there is a strong operation sop_p in between w_p^1 and r_q^1 in π''. Therefore, $w_p^1 \stackrel{fcs_p}{\Longrightarrow} sop_p$ and thus $w_p^1 \stackrel{\tau_i}{\longrightarrow} sop_p$. Therefore, sop_p is not in τ'_i since w_p^1 is not in τ'_i, and hence is weak, which is a contradiction. Therefore, $r_q^1(z, v_1)$ and $w_p^1(z, v_2)$ are not ordered by the happens before relation in π'', and hence, in σ. Thus, there exists a data race between $r_q^1(z, v_1)$ and $w_p^1(z, v_2)$ in σ. It follows that σ' contains a data race.

Now, σ' is a legal sequence of operations due to the change in the value returned by the read, implied by replacing r_q^1 with r_q^2. Since $r_q^2(z, v_2)$ is the last operation in σ', it clearly does not influence any operation in σ'. By Lemma 3.3, σ' is partially

admissible for some set of flow control sequences $\{fcs''_l\}_{l=1}^{n}$. Therefore, σ' is a prefix of sequential execution of `Prog` which contains a data race between $r_q^2(z, v_2)$ and $w_p^1(z, v_2)$, as needed. □

Hence, we have shown that if τ_i is not fully admissible, then there exists a prefix of a sequential execution of `Prog` which contains a data race. □

The proof of the following theorem is now immediate.

THEOREM 6.6. *Every hybrid consistent execution of a data-race-free program is sequentially consistent.*

Proof. Let R be a hybrid consistent execution of a data-race-free program `Prog`. Fix a subset S of the memory operations in R and a set of flow control sequences fcs_i from the definition of hybrid consistency. Choose a process p_i. Let τ_i be a minimal legal permutation of S as guaranteed by the definition of hybrid consistency. Since `Prog` is data-race-free, no sequential execution has a data race. Then by Lemma 6.2, τ_i is fully admissible. Thus, R is a sequentially consistent execution of `Prog`. □

6.3. Reordering process' operations. The original definition of hybrid consistency in [12] did not include the last property, i.e., that all writes by the same process to the same object appear in the views of all other processes in the order implied by the flow control sequence of their invoking process. (The original definition also did not consider control operations.) Thus two weak operations by the same process p_i accessing the same location could be viewed by other processes in a different order from the order in which they appear in fcs_i. We show that under this behavior it is not true that every hybrid consistent execution of a data-race-free program is sequentially consistent.

Consider the following data-race-free program, assuming x and y are initially 0 and all instructions are weak:

p_1's program	p_2's program
$\text{tmp}_1 := \text{read}(x)$;	$\text{tmp}_2 := \text{read}(y)$;
$\text{write}(x, 5)$;	$\text{write}(y, 5)$;
if $\text{tmp}_1 = 5$ then	if $\text{tmp}_2 = 5$ then
$\text{write}(y, 5)$;	$\text{write}(x, 5)$.

Consider the following execution R:

$$R|p_1 = r_1^1(x, 5), w_1^2(x, 5), w_1^3(y, 5),$$
$$R|p_2 = r_2^1(y, 5), w_2^2(y, 5), w_2^3(x, 5).$$

Although R is not sequentially consistent (we leave it to the reader to verify this), it is hybrid consistent:

$$\tau_1 = w_2^2(y, 5), r_2^1(y, 5), w_2^3(x, 5), r_1^1(x, 5), w_1^2(x, 5), w_1^3(y, 5),$$
$$\tau_2 = w_1^2(x, 5), r_1^1(x, 5), w_1^3(y, 5), r_2^1(y, 5), w_2^2(y, 5), w_2^3(x, 5).$$

7. Discussion. As the demand for powerful computers grows faster than the technology to develop new processors, the need for highly parallel multiprocessors increases. However, in order to fully utilize such machines, convenient paradigms for writing concurrent programs must be developed. These paradigms should allow the user to enjoy the same simple model of the world as in uniprocessors, without sacrificing the performance of the whole system. These two goals are somewhat contradictory. Recent results indicate that there is a tradeoff between the similarity of a distributed shared memory to real shared memory, and the efficiency of the hardware.

In this paper we have tried to bridge these two contradictory goals. We presented a general framework which encompasses the functionality of the compiler and the runtime system and models their interaction with the memory consistency system. Our framework allows the definition of known consistency conditions to be combined with implementations that exploit optimizations for reducing the latency of memory accesses. To the best of our knowledge, our definitions are unique in explicitly modeling the whole program, rather than just looking at the memory operations in isolation.

We also characterized requirements on programs that guarantee that they will behave on hybrid consistent memories as if they are sequentially consistent. This allows programmers to reason about certain classes of programs assuming sequential consistency, yet run them on more efficient hardware. The approaches we studied were (1) labeling all writes as strong and all reads as weak, (2) using the previous scheme to implement efficient mutual exclusion, and (3) running data-race-free programs. Note that the first two approaches do not necessarily yield data-race-free programs.

In this paper, we have assumed that the application program at each node is a serial code or, in other words, consists of a single thread of execution. We believe our framework and results can be naturally extended to support multithreading as well. We briefly outline one approach. Instead of a flow control sequence, define a *flow control (partial) order* based on two new control operations, *fork* and *join*. A fork operation splits the code into several parallel threads of execution and a join operation joins several parallel threads of execution into one thread of execution. Operations that belong to the same thread of execution are ordered as in a flow control sequence and after the previous fork operation (if there exists one). Operations that belong to different parallel threads are not ordered by the flow control order. All operations that belong to threads that were joined by a join operation are ordered before the join operation that joined their threads. The definitions of the consistency conditions remain the same, with the one exception that we use the flow control orders instead of the flow control sequences.

This work is part of an ongoing attempt to understand consistency conditions and their implications on programming, compiler design, and architecture. Much research is still needed before this goal can be met. While more efficient, fault-tolerant algorithms for implementing various consistency conditions still need to be developed, our paper takes a complementary approach; it provides a clean and formal framework for investigating systematic methods, rules, and compiler techniques to transform programs written for strong consistency conditions into correct programs for weaker consistency conditions.

Acknowledgments. We would like to thank Kourosh Gharachorloo, Phil Gibbons, Martha Kosa, and Michael Merritt for helpful comments. The anonymous referees made helpful comments which improved the presentation.

REFERENCES

[1] R. Acosta, J. Kjelstrup, and H. Torng, *An instruction issuing approach to enhancing performance in multiple functional unit processors*, IEEE Trans. Comput., C-35 (1986), pp. 815–828.

[2] S. Adve, *Designing Memory Consistency Models for Shared-Memory Multiprocessors*, Technical Report 1198, Ph.D. thesis, Computer Science Department, University of Wisconsin, Madison, WI, 1993.

[3] S. Adve and M. Hill, *Weak ordering—a new definition*, in Proc. of the 17th International Symposium on Computer Architecture, ACM, New York, 1990, pp. 2–14.

[4] S. Adve and M. Hill, *A Unified Formalization of Four Shared-Memory Models*, Technical Report 1051, Computer Science Department, University of Wisconsin, Madison, WI, 1991.

[5] S. Adve and M. Hill, *Sufficient Conditions for Implementing the Data-Race-Free-*1 *Memory Model*, Technical Report 1107, Computer Science Department, University of Wisconsin, Madison, WI, 1992.

[6] S. Adve, M. Hill, B. Miller, and R. Netzer, *Detecting data races on weak memory systems*, in Proc. of the 18th International Symposium on Computer Architecture, ACM, New York, 1991, pp. 234–243.

[7] Y. Afek, G. Brown, and M. Merritt, *A lazy cache algorithm*, ACM Trans. Programming Lang. Systems, 15 (1993), pp. 182–205.

[8] M. Ahamad, R. Bazzi, R. John, P. Kohli, and G. Neiger, *The power of processor consistency*, in Proc. of the 5th ACM Symposium On Parallel Algorithms and Architectures, ACM, New York, 1993, pp. 251–260.

[9] M. Ahamad, J. Burns, P. Hutto, and G. Neiger, *Causal Memory*, in Proc. 5th International Workshop on Distributed Algorithms, Greece, 1991, Lecture Notes in Comput. Sci. 579, Springer-Verlag, New York, pp. 9–30.

[10] M. Ahamad, G. Neiger, P. Kohli, J. Burns, and P. Hutto, *Causal memory: Definitions, implementation, and programming*, Distrib. Comput., 9 (1993), pp. 37–49.

[11] H. Attiya, S. Chaudhuri, R. Friedman, and J. Welch, *Shared Memory Consistency Conditions for Nonsequential Execution: Definitions and Programming Strategies*, Technical Report LPCR 9306, Department of Computer Science, Technion, Israel, 1993.

[12] H. Attiya and R. Friedman, *A Correctness Condition for High-Performance Multiprocessors*, Technical Report 767, Department of Computer Science, Technion, Israel. SIAM J. Comput., 27 (1998), to appear.

[13] H. Attiya and R. Friedman, *Programming DEC-alpha based multiprocessors the easy way*, in Proc. 6th ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, 1994, pp. 157–166. Technical Report LPCR 9411, Department of Computer Science, Technion, Israel.

[14] H. Attiya and J. Welch, *Sequential consistency versus linearizability*, ACM Trans. Comput. Systems, 12 (1994), pp. 91–122.

[15] R. Bisiani, A. Nowatzyk, and M. Ravishankar, *Coherent shared memory on a distributed memory machine*, in Proc. International Conference on Parallel Processing, Penn. State Univ. Press, Philadelphia, PA, 1989, pp. I–133–141.

[16] J.-D. Choi and S. L. Min, *Race frontier: Reproducing data races in parallel program debugging*, in Proc. of the 3rd ACM Symposium on Principles and Practice of Parallel Programming, ACM, New York, 1991, pp. 145–154.

[17] A. Dinning and E. Schonberg, *Detecting access anomalies in programs with critical sections*, in Proc. of the ACM Workshop on Parallel and Distributed Debugging, ACM, New York, 1991, pp. 85–96.

[18] M. Dubois and C. Scheurich, *Memory access dependencies in shared-memory multiprocessors*, IEEE Trans. Software Engrg., 16 (1990), pp. 660–673.

[19] M. Dubois, C. Scheurich, and F. A. Briggs, *Synchronization, coherence and event ordering in multiprocessors*, IEEE Comput., 21 (1988), pp. 9–21.

[20] J. Fisher, *Very long instruction word architectures and the ELI-*512, in Proc. of the 10th International Symposium on Computer Architecture, ACM, New York, 1991, pp. 140–150.

[21] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy, *Memory consistency and event ordering in scalable shared-memory multiprocessors*, in Proc. of the 17th International Symposium on Computer Architecture, ACM, New York, 1990, pp. 15–26.

[22] P. Gibbons and M. Merritt, *Specifying non-blocking shared memories*, in Proc. of the 4th ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, 1992, pp. 306–315.

[23] P. Gibbons, M. Merritt, and K. Gharachorloo, *Proving sequential consistency of high-performance shared memories*, in Proc. of the 3rd ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, 1991, pp. 292–303.

[24] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers Inc., San Mateo, CA, 1990, pp. 251–349.

[25] M. Herlihy and J. Wing, *Linearizability: A correctness condition for concurrent objects*, ACM Trans. Programming Lang. Systems, 12 (1990), pp. 463–492.

[26] M. S. Lam and R. P. Wilson, *Limits of control flow on parallelism*, in Proc. of the 19th International Symposium on Computer Architecture, ACM, New York, 1992, pp. 46–57.

[27] L. Lamport, *Time, clocks and the ordering of event in a distributed system*, Comm. ACM, 21 (1978), pp. 558–565.

[28] L. Lamport, *How to make a multiprocessor computer that correctly executes multiprocess programs*, IEEE Trans. Comput., C-28 (1979), pp. 690–691.

[29] R. Lipton and J. Sandberg, *PRAM: A Scalable Shared Memory*, Technical Report CS-TR-180-88, Computer Science Department, Princeton University, Princeton, NJ, 1988.

[30] J. Mellor-Crummey, *On-the-fly detection of data races for programs with nested fork-join parallelism*, Proc. Supercomputer Debugging Workshop, Los Alamitos Natl. Lab, Albuquerque, NM, 1991, pp. 24–33.

[31] R. Netzer, *Race Condition Detection for Debugging Shared-Memory Parallel Programs*, Technical Report 1039, Ph.D. thesis, Computer Science Department, University of Wisconsin, Madison, WI, 1991.

[32] R. Netzer and B. Miller, *Improving the accuracy of data race detection*, in Proc. of the 3rd ACM Symposium on Principles and Practice of Parallel Programming, ACM, New York, 1991, pp. 133–144.

[33] R. Netzer and B. Miller, *What are race conditions? Some issues and formalizations*, ACM Lett. Programming Lang. Systems, 1 (1992), pp. 74–88.

[34] Y. Patt, W. Hwu, and M. Shebanow, *HPS, a new microarchitecture: Rationale and introduction*, in Proc. of the 18th Annual Microprogramming Workshop, ACM, New York, 1985, pp. 103–108.

[35] A. Peleg and U. Weiser, *Future trends in microprocessors: Out-of-order execution, speculative branching and their CISC performance models*, in Proc. of the 17th Convention of Electrical and Electronics Engineers in Israel, IEEE Computer Society Press, Los Alamitos, CA, 1991, pp. 263–266.

[36] M. Raynal, *Algorithms for Mutual Exclusion*, MIT Press, Cambridge, MA, 1986.

[37] D. Shasha and M. Snir, *Correct and efficient execution of parallel programs that share memory*, ACM Trans. Programming Lang. Systems, 10 (1988), pp. 282–312.

[38] A. Singh, *A Framework for Programming Using Non-Atomic Variables*, Technical Report TRCS-93-11, Department of Computer Science, University of California at Santa Barbara, 1993.

[39] B. Smith, *A massively parallel shared memory computer*, in 3rd ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, 1991, p. 123.

[40] J. Smith, *Dynamic instruction scheduling and the astronautics ZS*-1, IEEE Comput., 22 (1989), pp. 21–35.

[41] R. N. Zucker and J.-L. Baer, *A performance study of memory consistency models*, in Proc. of the 19th International Symposium on Computer Architecture, ACM, New York, 1992, pp. 2–12.

SIAM J. COMPUT.
Vol. 27, No. 1, pp. 90–106, February 1998
© 1998 Society for Industrial and Applied Mathematics
004

TWO-DIMENSIONAL PERIODICITY IN RECTANGULAR ARRAYS*

AMIHOOD AMIR† AND GARY BENSON‡

Abstract. String matching is rich with a variety of algorithmic tools. In contrast, multidimensional matching has had a rather sparse set of techniques. This paper presents a new algorithmic technique for two-dimensional matching: *periodicity analysis*. Its strength appears to lie in the fact that it is inherently two-dimensional.

Periodicity in strings has been used to solve string matching problems. Multidimensional periodicity, however, is not as simple as it is in strings and was not formally studied or used in pattern matching. In this paper, we define and analyze two-dimensional periodicity in rectangular arrays. One definition of string periodicity is that a periodic string can self-overlap in a particular way. An analogous concept is true in two dimensions. The *self-overlap vectors* of a rectangle generate a regular pattern of locations where the rectangle may originate. Based on this regularity, we define four categories of periodic arrays—*nonperiodic, lattice periodic, line periodic, and radiant periodic*—and prove theorems about the properties of the classes.

We give serial and parallel algorithms that find all locations where an overlap originates. In addition, our algorithms find a *witness* proving that the array does not self-overlap in any other location. The serial algorithm runs in time $O(m^2)$ (linear time) when the alphabet size is finite, and in $O(m^2 \log m)$ otherwise. The parallel algorithm runs in time $O(\log m)$ using $O(m^2)$ CRCW processors.

Key words. string matching, two-dimensional, periodicity, witness, sequential algorithm, parallel algorithm

AMS subject classification. 68Q25

PII. S0097539795298321

1. Introduction.

String matching is a field rich with a variety of algorithmic ideas. The early string matching algorithms were mostly based on constructing a pattern automaton and subsequently using it to find all pattern appearances in a given text ([KMP-77, AC-75, BM-77]). Recently developed algorithms [G-85, V-85, V-91] use periodicity in strings to solve this classic string matching problem. Lately, there has been interest in various two-dimensional approximate matching problems, largely motivated by low-level image processing ([KS-87, AL-91, AF-91, ALV-90]). Unlike string matching, the methods for solving multidimensional matching problems are scant. This paper adds a *new algorithmic tool* to the rather empty tool chest of multidimensional matching techniques: two-dimensional periodicity analysis.

String periodicity is an intuitively clear concept and the properties of a string period are simple and well understood. Two-dimensional periodicity, however, presents some difficulties. Periodicity in the *plane* is easy to define. However, we seek the period of a *finite rectangle*. We have chosen to concentrate on a periodicity definition that implies the ability for *self-overlap*. In strings, such an overlap allows definition of

* Received by the editors April 17, 1995; accepted for publication (in revised form) December 19, 1995. A preliminary version of this paper was presented at the 3rd ACM-SIAM Symposium on Discrete Algorithms, Orlando, FL, SIAM, Philadelphia, 1992, pp. 440–452. This research was partially supported by NSF grant IRI-9013055.

http://www.siam.org/journals/sicomp/27-1/29832.html

† College of Computing, Georgia Institute of Technology, Atlanta, GA 30332 (amir@gatech.edu). This author was partially supported by NSF grant CCR-95-31939 and the Israel Ministry of Science and the Arts grants 6297 and 8560.

‡ Department of Biomathematical Sciences, Mount Sinai Medical Center, New York, NY 10029 (benson@ecology.biomath.mssm.edu). This author was partially supported by NSF grant CCR-9623532.

a smallest period whose concatenation produces the entire string. The main contribution of this paper is showing that for rectangles also, the overlap produces a "smallest unit" and a regular pattern in which it appears in the array. The main differences are that this "smallest unit" is a vector rather than a subblock of the array, and that the pattern is not a simple concatenation. Rather, based on the patterns of vectors that can occur, there are four categories of array periodicity: *nonperiodic, line periodic, radiant periodic*, and *lattice periodic*. As in string matching, this regularity can be exploited.

The strength of periodicity analysis appears to lie in the fact that it is inherently a two-dimensional technique, whereas most previous work on two-dimensional matching has reduced the matrix problem to a problem on strings and then applied one-dimensional string matching methods. The two-dimensional periodicity analysis has already proven useful in solving several multidimensional matching problems [ABF-94, ABF-93, ABF-97, KR-94]. We illustrate with two examples.

The original motivation for this work was our research in image preserving compression. We wanted to solve the following problem: given a two-dimensional pattern P and a two-dimensional text T which has been compressed, find all occurrences of P in T without decompressing the text. The goal is a sublinear algorithm with respect to the size of the original *uncompressed* text. Some initial success in this problem was achieved in [ALV-90], but those authors' algorithm, being automaton based, seems to require a large amount of decompression. In [AB-92b, ABF-97], we used periodicity to find the first *optimal* pattern matching algorithm for compressed two-dimensional texts.

Another application is the two-dimensional exact matching problem. Here the text is not compressed. Baker [B-78] and, independently, Bird [Bi-77] used the Aho and Corasick [AC-75] dictionary matching algorithm to obtain a $O(n^2 \log |\Sigma|)$ algorithm for this problem. This algorithm is automaton based, and therefore the running time of the text scanning phase is dependent on the size of the alphabet. In [ABF-94] we used periodicity analysis to produce the first two-dimensional exact matching algorithm with a linear time *alphabet independent* text scanning phase.

Since the work presented here first appeared [AB-92a], the analysis of radiant-periodic patterns has been strengthened [GP-92, RR-93], and periodicity analysis has additionally proven useful in providing optimal parallel two-dimensional matching algorithms [ABF-93, CCG+93], as well as in solving a three-dimensional matching problem [KR-94].

This paper is organized as follows. In section 2, we review periodicity in strings and extend this notion to two dimensions. In section 3, we give formal definitions, describe the classification scheme for the four types of two-dimensional periodicity, and prove some theorems about the properties of the classes. In section 4 we present serial and parallel algorithms for detecting the type of periodicity in an array. The complexity of the serial algorithm is $O(m^2)$ (linear time) when the alphabet size is finite, and $O(m^2 \log m)$ otherwise. The parallel algorithm runs in time $O(\log m)$ with $O(m^2)$ CRCW processors. In addition to knowing where an array can self-overlap, knowing where it cannot and why is also useful. If an overlap is not possible, then the overlap produces some mismatch. Our algorithms find a single mismatch location or *witness* for each self-overlap that fails.

2. Periodicity in strings and arrays. In a periodic string, a smallest period can be found whose concatenation generates the entire string. In two dimensions, if an array were to extend infinitely so as to cover the plane, the one-dimensional notion

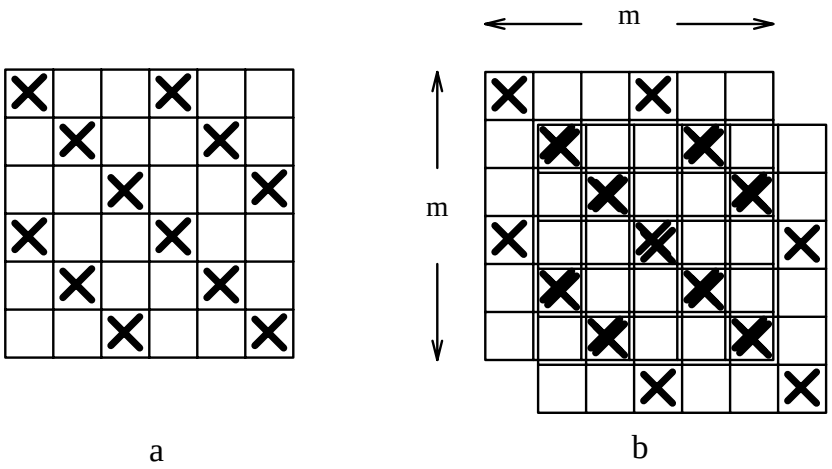


FIG. 1. (a) *A periodic pattern.* (b) *A suffix matches a prefix.*

of a period could be generalized to a unit cell of a lattice. But a rectangular array is not infinite and may cut a unit cell in many different ways at its edges.

Instead of defining two-dimensional periodicity on the basis of some subunit of the array, we instead use the idea of *self-overlap*. This idea applies also to strings. A string w is *periodic* if the longest prefix p of w that is also a suffix of w is at least half the length of w. For example, if $w = abcabcabcabcab$, then $p = abcabcabcab$, and since p is over half as long as w, w is periodic. This definition implies that w may overlap itself starting in the fourth position.

The preceding idea easily generalized to two dimensions as illustrated in Figure 1.

DEFINITION 2.1. *Let A be a two-dimensional array. Call a* prefix *of A a rectangular subarray that contains one corner of A. (In the figure, the upper left corner.) Call a* suffix *of A a rectangular subarray that contains the diagonally opposite corner of A (in the figure, the lower right corner). We say A is* periodic *if the largest prefix that is also a suffix has dimensions at least as large as some fixed percentage d of the dimensions of A.*

In the figure, if $d \leq \frac{5}{6}$, then A is periodic. As with strings, if A is periodic, then A may overlap itself if the prefix of one copy of A is aligned with the suffix of a second copy of A. Notice that both the upper left and lower left corners of A can define prefixes, giving A two directions in which it can be periodic. As we will describe in the next section, the classification of periodicity type for A is based on whether it is periodic in either or both of these directions.

3. Classifying arrays. Our goal here is classifying an array A into one of four periodicity classes. For clarity of presentation we concentrate on square arrays. We later show how to generalize all results to rectangles. We begin with some definitions of two-dimensional periodicity and related concepts (Figure 2).

DEFINITION 3.1. *Let $A[0 \ldots m-1, 0 \ldots m-1]$ be an $m \times m$ square array. Each element of A contains a symbol from an alphabet Σ. A subarray of A is called a block. Blocks are designated by their first and last row and column. Thus, the block $A[0 \ldots m-1, 0 \ldots m-1]$ is the entire array. Each corner of A defines a quadrant. Quadrants are labeled counterclockwise from upper left, quadrants* I, II, III, *and* IV.

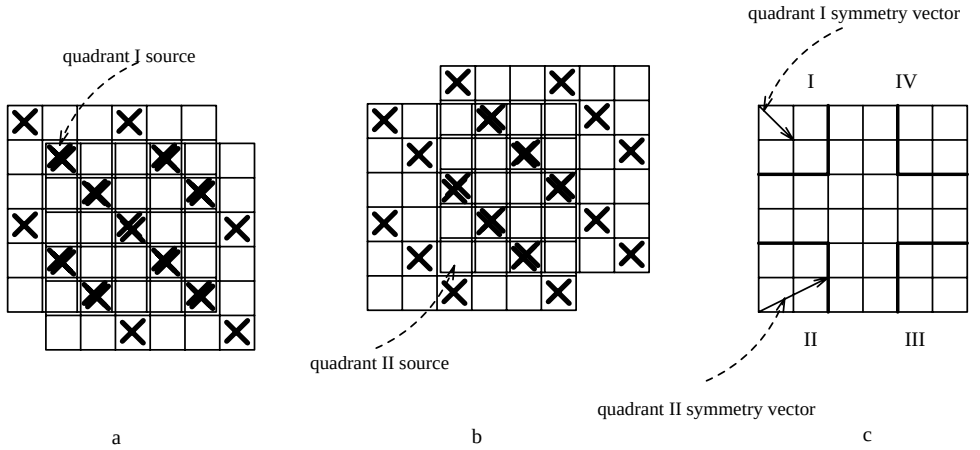


FIG. 2. *Two overlapping copies of the same array.* (a) *A quadrant* I *source.* (b) *A quadrant* II *source.* (c) *The symmetry vectors.*

Each quadrant has size q*, where* $1 \le q \le \lceil \frac{m}{2} \rceil$*. (Quadrants may share part of a row or column.) Quadrant* I *is the block* $A[0 \dots q-1, 0 \dots q-1]$*. The choice of* q *may depend on the application. For this paper,* $q = \lceil \frac{m}{3} \rceil$*.*

DEFINITION 3.2. *Suppose we have two copies of* A*, one directly on top of the other. The copies are said to be* in register *because some of the elements overlap (in this case, all the elements) and overlapping elements contain the same symbol. If the two copies can be repositioned so that* $A[0,0]$ *overlaps* $A[r,c]$ $(r \ge 0, c > 0)$ *and the copies are again in register, then we say that the array is* quadrant I symmetric*, that* $A[r,c]$ *is a* quadrant I source*, and that vector* $\vec{v} = r\vec{y} + c\vec{x}$ *is a* quadrant I symmetry vector*. Here,* $\vec{y}$ *is the vertical unit vector in the direction of increasing row index and* $\vec{x}$ *is the horizontal unit vector in the direction of increasing column index. If the two copies can be repositioned so that* $A[m-1,0]$ *overlaps* $A[r,c]$ $(r < m-1, c \ge 0)$ *and the copies are again in register, then we say that the array is* quadrant II symmetric*, that* $A[r,c]$ *is a* quadrant II source*, and that* $\vec{v} = (r-m+1)\vec{y} + c\vec{x}$ *is a* quadrant II symmetry vector*.*

Analogous definitions exist for quadrants III and IV, but by symmetry, if $\vec{v}$ is a quadrant III(IV) symmetry vector, then $-\vec{v}$ is a quadrant I(II) symmetry vector. We will usually indicate a vector $\vec{v} = r\vec{y} + c\vec{x}$ by the ordered pair (r,c). Note that symmetry vector (r,c) defines a mapping between identical elements, that is, (r,c) is a symmetry vector iff $A[i,j] = A[i+r, j+c]$ wherever both elements are defined. In particular, if (r,c) is a symmetry vector, then it maps the block $A[i \dots j, k \dots l]$ to the identical block $A[i+r \dots j+r, k+c \dots l+c]$.

In the remainder of this paper, we use the terms *source* and *symmetry vector* interchangeably.

DEFINITION 3.3. *The* length *of a symmetry vector is the maximum of the absolute values of its coefficients. A* lexicographic ordering *of quadrant* I *vectors (quadrant* II *vectors) is accomplished by sorting the vectors first by length, and then, for vectors of the same length, by reverse sorting them by column coefficient and then sorting them by row coefficient (by reverse sorting them by absolute value of row coefficient and then sorting them by column coefficient). A* shortest *vector is the smallest in its lexicographic ordering. The* basis vectors *for array* A *consist of the shortest quadrant*

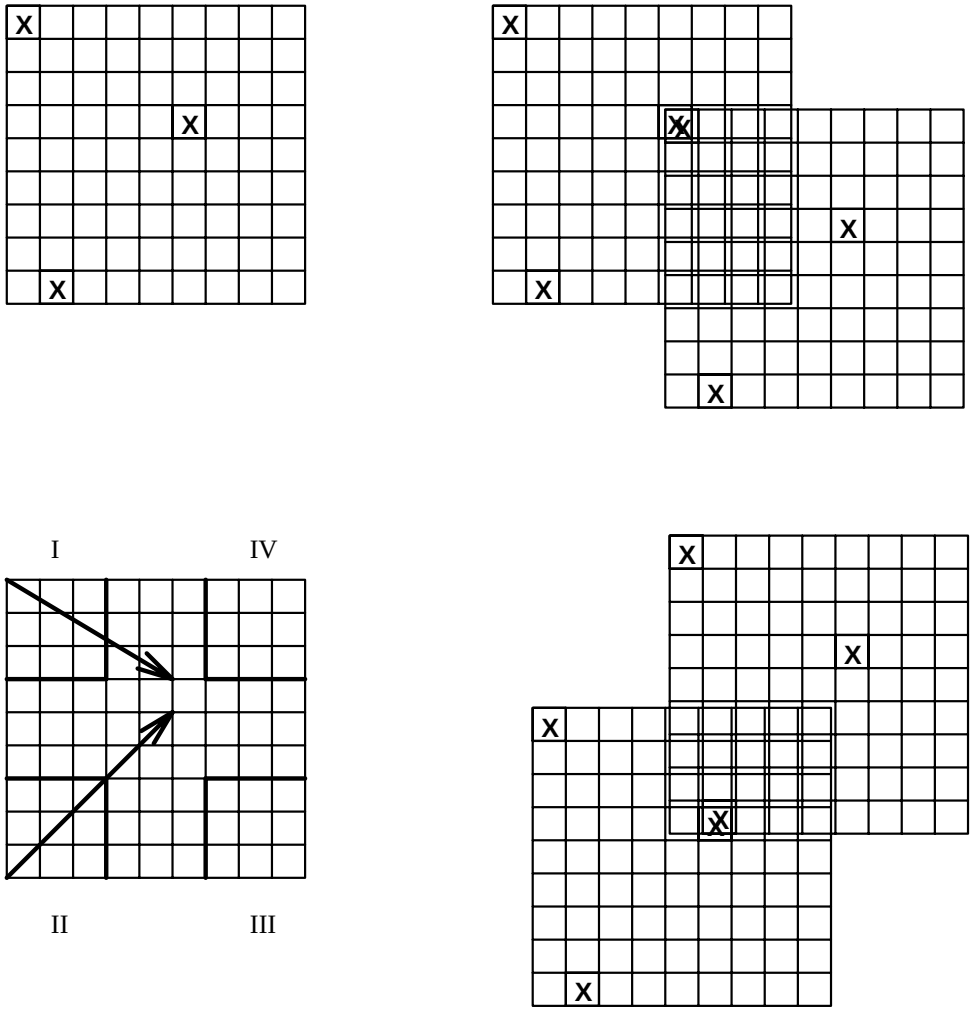


FIG. 3. *Nonperiodic array.*

I *vector* (r_1, c_1) *(if any) and the shortest quadrant* II *vector* (r_2, c_2) *(if any). If the length of a symmetry vector is* $< p$ *where* $p = \lceil \frac{m}{3} \rceil$ *then the vector is* periodic.

We are now ready to classify a square array A into one of four periodicity classes based on the presence or absence of periodic vectors in quadrants I and II. Following the classification, we prove some theorems about the properties of the classes. In section 4 we present algorithms for finding *all* the sources in an array.

The four classes of two-dimensional periodicity are (Figures 3–6)

- *nonperiodic* — the array has no periodic vectors.
- *lattice periodic* — the array has periodic vectors in both quadrants. All quadrant I sources which occur in quadrant I fall on the nodes of a lattice which is defined by the basis vectors. The same is true for quadrant II sources in quadrant II. Specifically, let $\vec{v}_1 = (r_1, c_1)$ and $\vec{v}_2 = (r_2, c_2)$ be the periodic basis vectors in quadrants I and II, respectively. Then, an element in quadrant I is a quadrant I source iff it occurs at index $A[ir_1 + jr_2, ic_1 + jc_2]$ for integers i, j. An element in quadrant II is a quadrant II source iff it occurs at index

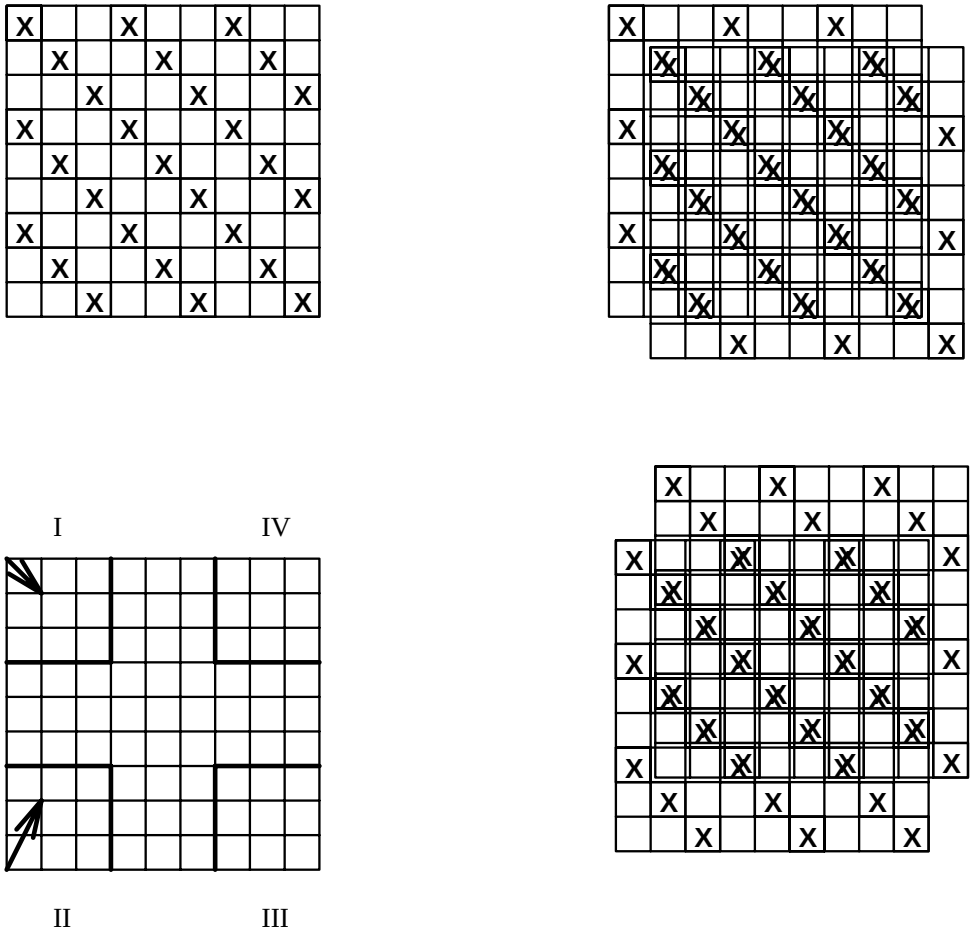


FIG. 4. *Lattice-periodic array.*

$A[m-1+\hat{\imath}r_1+\hat{\jmath}r_2, \hat{\imath}c_1+\hat{\jmath}c_2]$ for integers $\hat{\imath}, \hat{\jmath}$.

- *line periodic* — the array has a periodic vector in only one quadrant and the sources in that quadrant all fall on one line.
- *radiant periodic* — this category is identical to the line periodic category, except that in the quadrant with the periodic vector, the sources fall on several lines which all radiate from the quadrant's corner. We do not describe the exact location of the sources for this class; see [GP-92] for a detailed analysis of the source locations.

Next, we prove some theorems about the properties of the classes. All the theorems are stated in terms of square arrays for clarity. At the end of the theorems we explain how they can be modified to apply to any $n \times m$ rectangular array.

In Lemmas 3.4–3.6, we establish the fact that if we have symmetry vectors for both quadrants I and II, and they meet a pair of constraints on the sum of their coefficients, then every linear combination of the vectors defines another symmetry vector.

LEMMA 3.4. *If* (r_1, c_1) *and* (r_2, c_2) *are symmetry vectors from quadrants* I *and* II*, respectively, and* $c_1 + c_2 < m$ *and* $r_1 + |r_2| < m$*, then* $(r_1 + r_2, c_1 + c_2)$ *is either a*

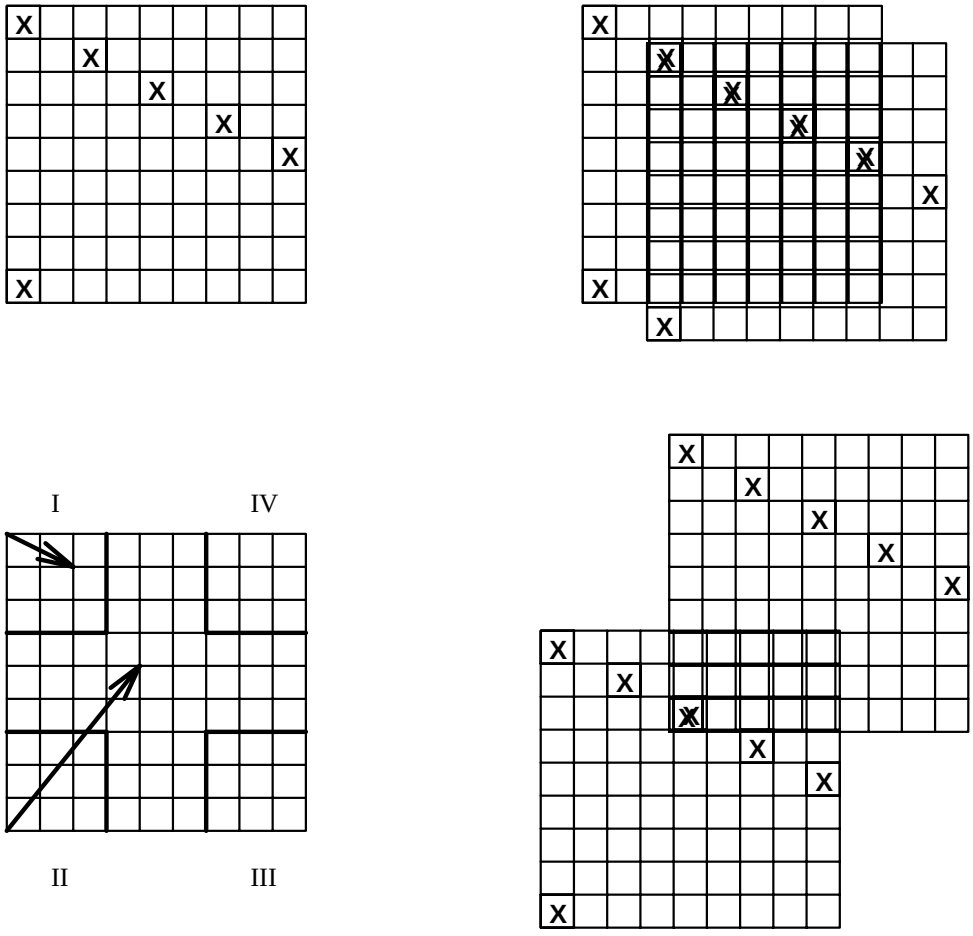


FIG. 5. *Line-periodic array.*

quadrant I *symmetry vector* $(r_1 \geq |r_2|)$ *or a quadrant* II *symmetry vector* $(r_1 < |r_2|)$.

Proof. We prove for the case $r_1 \geq |r_2|$. The proof for the other case is similar. We show that $S = (r_1 + r_2, c_1 + c_2)$ is a quadrant I source. First, by the constraint on the c_i, the fact that r_2 is negative and the assumption that $r_1 \geq |r_2|$, S is an element of A. Next, we show via two pairs of mappings that the quadrant I prefix block $A[0 \ldots m - r_1 - r_2 - 1, 0 \ldots m - c_1 - c_2 - 1]$ is identical to the suffix block $A[r_1 + r_2 \ldots m - 1, c_1 + c_2 \ldots m - 1]$.

First pair: (r_1, c_1) maps block $A[0 \ldots m - r_1 - 1, 0 \ldots m - c_1 - c_2 - 1]$ to block $A[r_1 \ldots m - 1, c_1 \ldots m - c_2 - 1]$. (r_2, c_2) maps the resultant block to block $A[r_1 + r_2 \ldots m + r_2 - 1, c_1 + c_2 \ldots m - 1]$.

Second pair: (r_2, c_2) maps block $A[m - r_1 \ldots m - r_1 - r_2 - 1, 0 \ldots m - c_1 - c_2 - 1]$ to $A[m - r_1 + r_2 \ldots m - r_1 - 1, c_1 \ldots m - c_2 - 1]$. (r_1, c_1) maps the resultant block to $A[m + r_2 \ldots m - 1, c_1 + c_2 \ldots m - 1]$. □

LEMMA 3.5. *If* (r_1, c_1) *and* (r_2, c_2) *are two quadrant* k *symmetry vectors* ($k =$ I, II) *and* $|r_1| + |r_2| < m$ *and* $c_1 + c_2 < m$, *then* $(r_1 + r_2, c_1 + c_2)$ *is also a quadrant* k *symmetry vector.*

Proof. We prove for quadrant I. The proof for the other quadrant is similar. We

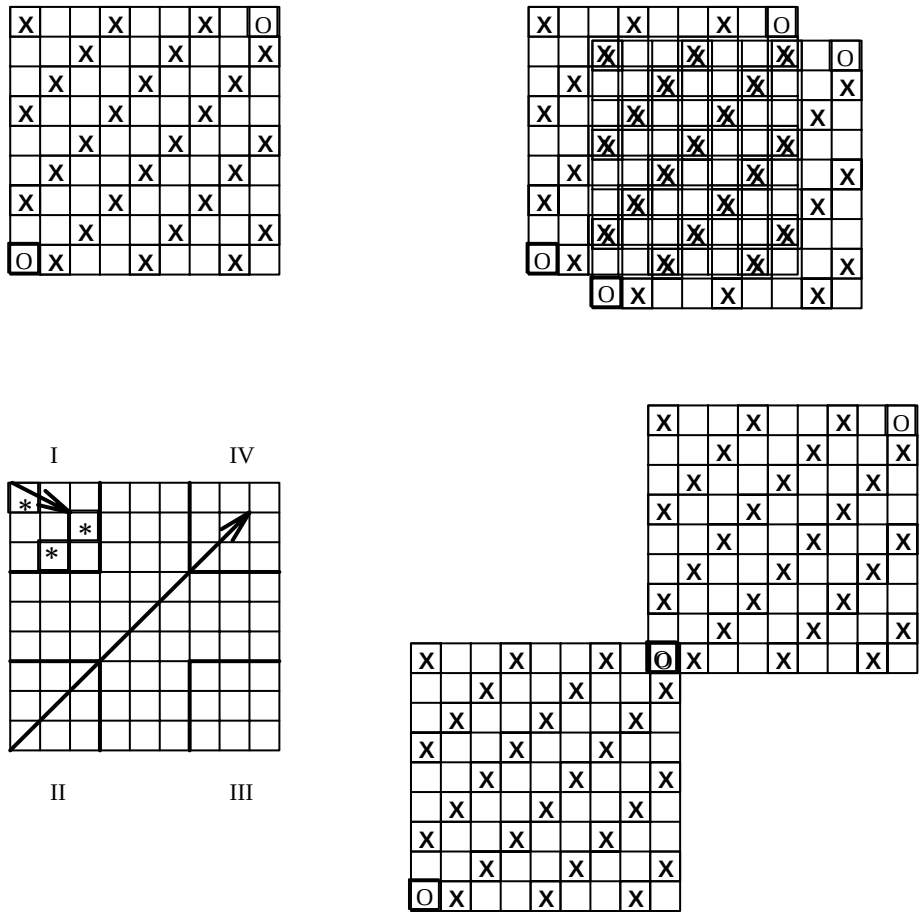


FIG. 6. *Radiant-periodic array. Three noncolinear sources are starred.*

show that $S = (r_1 + r_2, c_1 + c_2)$ is a quadrant I source. First, by the restraints on the r_i and the c_i, S is an element of A. Next, by a pair of mappings, we show that the quadrant I prefix block $A[0 \ldots m - r_1 - r_2 - 1, 0 \ldots m - c_1 - c_2 - 1]$ is identical to the suffix block $A[r_1 + r_2 \ldots m - 1, c_1 + c_2 \ldots m - 1]$. Recall that both r_1 and r_2 are positive.

First mapping: (r_1, c_1) maps the block $A[0 \ldots m - r_1 - r_2 - 1, 0 \ldots m - c_1 - c_2 - 1]$ to the block $A[r_1 \ldots m - r_2 - 1, c_1 \ldots m - c_2 - 1]$. Second mapping: (r_2, c_2) maps the resultant block to the block $A[r_1 + r_2 \ldots m - 1, c_1 + c_2 \ldots m - 1]$. □

LEMMA 3.6. *If* $\vec{v}_1 = (r_1, c_1)$ *and* $\vec{v}_2 = (r_2, c_2)$ *are symmetry vectors from quadrants* I *and* II*, respectively, and* $c_1 + c_2 < m$ *and* $r_1 + |r_2| < m$*, then for all integers* i, j *such that* $A[ir_1 + jr_2, ic_1 + jc_2]$ *is an element of* A*,* $(ir_1 + jr_2, ic_1 + jc_2)$ *is a quadrant* I *symmetry vector. Similarly, for all* $\hat{\imath}, \hat{\jmath}$ *such that* $A[m - 1 + \hat{\imath}r_1 + \hat{\jmath}r_2, \hat{\imath}c_1 + \hat{\jmath}c_2]$ *is an element of* A*,* $(\hat{\imath}r_1 + \hat{\jmath}r_2, \hat{\imath}c_1 + \hat{\jmath}c_2)$ *is a quadrant* II *symmetry vector.*

Proof. We prove for vector $(ir_1 + jr_2, ic_1 + jc_2)$, equivalent to source $S_{i,j} = A[ir_1 + jr_2, ic_1 + jc_2]$. The proof for the other vector is similar. Consider the lattice of elements in A defined by the quadrant I and II vectors and with one element at

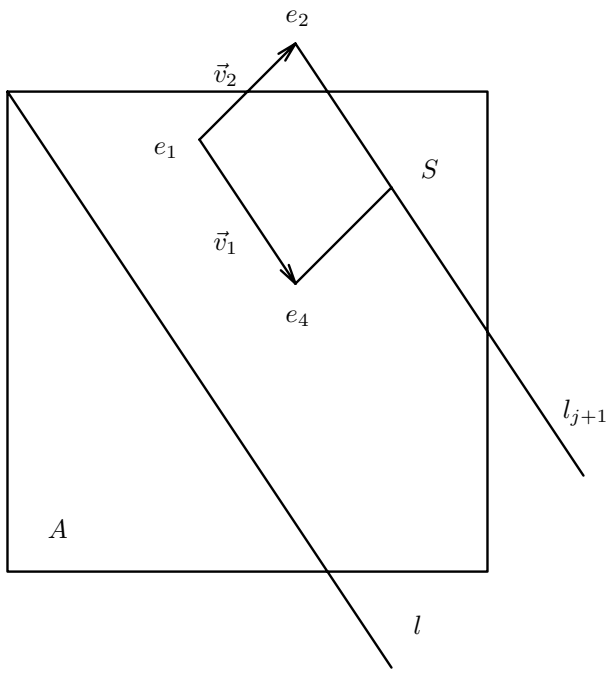


FIG. 7. *A candidate source S in Lemma* 3.6. *Here* $|r_1| \geq |r_2|$.

$A[0,0]$. (The lattice elements correspond exactly to the elements $S_{i,j}$.) Consider the line l that extends from element $A[0,0]$ through elements $S_{i,0} = A[ir_1, ic_1]$. We prove the lemma only for those lattice elements on or to the right of l. The remaining elements are treated similarly.

Case 1. $S_{i,0}$ is on line l. For $S_{1,0}$, $i = 1$ and (r_1, c_1) is a symmetry vector by hypothesis. Now, by induction on i, assume that (ir_1, ic_1) is a symmetry vector. Since (r_1, c_1) and (ir_1, ic_1) are both quadrant I symmetry vectors, by Lemma 3.5 $((i+1)r_1, (i+1)c_1)$ is a quadrant I symmetry vector.

Case 2. $S_{i,j}$ $j \geq 1$ to the right of line l. Elements $S_{i,j}$ fall on lines l_j which are parallel to line l. We show that the *uppermost* element $S_{i,j}$ is a source. By application of Lemma 3.5, as in Case 1, the remaining sources on l_j are established.

Consider a *cell* of the lattice with sides (r_1, c_1) and (r_2, c_2) and corners

$$\begin{aligned} e_1 &= A[ir_1 + jr_2, ic_1 + jc_2], \\ e_2 &= A[ir_1 + (j+1)r_2, ic_1 + (j+1)c_2], \\ e_4 &= A[(i+1)r_1 + jr_2, (i+1)c_1 + jc_2], \\ S &= A[(i+1)r_1 + (j+1)r_2, (i+1)c_1 + (j+1)c_2], \end{aligned}$$

where S is the uppermost lattice element on line l_{j+1} (Figure 7). The following are always true:

- e_2 is **not** an element of A. Otherwise e_2—not S—is the top element on its line.
- e_4 **is** an element of A. Otherwise S is not in A, S is not to the right of line l or $r_1 + |r_2| \geq m$.

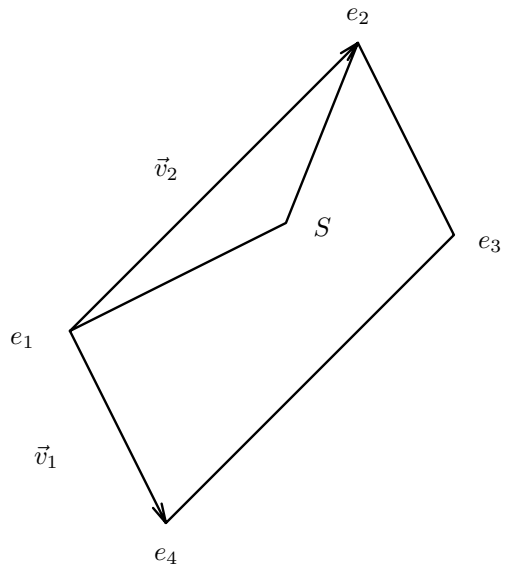


FIG. 8. *One of the vectors from e_1 to S or S to e_2 is a quadrant* II *vector shorter than $\vec{v}_2$.*

Two possibilities remain. Either e_1 is an element of A or it is not. Our proof is by induction on i and j. For the base cases we use $\vec{v}_1$ $(i = 1, j = 0)$, $\vec{v}_2$ $(i = 0, j = 1)$, and $\vec{v}_3 = \vec{v}_1 + \vec{v}_2$ which is either a quadrant I vector $(r_1 \geq |r_2|)$ or a quadrant II vector $(r_1 < |r_2|)$ by Lemma 3.4.

Subcase A. $r_1 \geq |r_2|$.

- e_1 is not an element of A. By the induction hypothesis, $\vec{v}_{e_4} = (i+1)\vec{v}_1 + j\vec{v}_2$ is a symmetry vector. Since e_1 is not on A, $r_{e_4} < r_1$. That is, the row coefficient in $\vec{v}_{e_4}$ is smaller than the row coefficient in $\vec{v}_1$. Apply Lemma 3.4 to $\vec{v}_{e_4}$ and $\vec{v}_2$ and S is a source.
- e_1 is an element of A. By the induction hypothesis, $\vec{v}_{e_1} = i\vec{v}_1 + j\vec{v}_2$ is a quadrant I symmetry vector. From the base case, $\vec{v}_3 = \vec{v}_1 + \vec{v}_2$ is a quadrant I symmetry vector. Apply Lemma 3.5 to $\vec{v}_{e_1}$ and $\vec{v}_3$ and S is a source.

Subcase B. $r_1 < |r_2|$.

- e_1 is not an element of A. It is impossible, else S is not in A or S is not right of l.
- e_1 is an element of A. Note that S is above row r_1 or else e_2 is on the array. The vector $\vec{v}_{e_2} = i\vec{v}_1 + (j+1)\vec{v}_2$ is a quadrant II symmetry vector (because r_{e_2} is negative) by application of Subcase A to quadrant II. Now, $r_{e_2} + r_1 =$(the row index of S)≥ 0, so $r_1 \geq -r_{e_2}$ or $|r_{e_2}| \leq r_1$. By hypothesis, $r_1 < |r_2|$ and therefore $|r_{e_2}| < |r_2|$. Apply Lemma 3.5 to $\vec{v}_1$ and $\vec{v}_{e_2}$ and S is a source. □

The proof of Theorem 3.9 is simplified by the following easily proven observation.

OBSERVATION 3.7. *Let (r_1, c_1) and (r_2, c_2) be symmetry vectors from quadrants* I *and* II*, respectively, and $c_1 + c_2 < m$ and $r_1 + |r_2| < m$, and let L be an infinite lattice of points on the xy-plane also with basis vectors (r_1, c_1) and (r_2, c_2). If we put one copy of A on each lattice point by aligning element $A[0, 0]$ with the lattice point, then the copies are in register and completely cover the plane.*

The next lemma establishes that for a given lattice of elements in A, an element

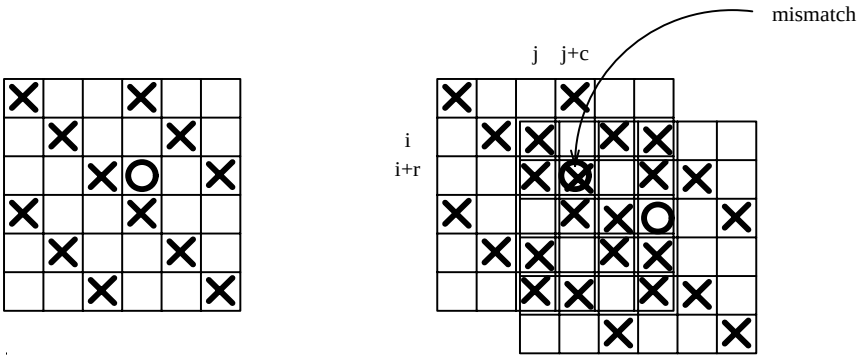


FIG. 9. *The witness table gives the location of a mismatch (if one exists) for two overlapping patterns:* $Witness[i, j] = [r, c]$.

not on the lattice has a shorter vector to some lattice point than the corresponding basis vector for the lattice. (Note that a simplified version of the proof appeared in [GP-92], and we use essentially that same proof here.)

LEMMA 3.8. *Let L be an infinite lattice in the xy-plane with basis vectors $\vec{v}_1 = (r_1, c_1)$ and $\vec{v}_2 = (r_2, c_2)$ (quadrants* I *and* II *symmetry vectors, respectively), where all the r_i and c_i are integers. Then, for any point $S = (x, y)$ that is not a lattice element, where x and y are integers, there exists a lattice point e such that the vector $\vec{v}$ from e to S (or S to e) is a quadrant* I *vector shorter than $\vec{v}_1$ or a quadrant* II *vector shorter than $\vec{v}_2$.*

Proof. Let S be an element that does not fall on a lattice point. Consider the unit cell of the lattice containing S (Figure 8) with nodes labeled e_1, e_2, e_3, and e_4, where

$$\begin{aligned} e_4 &= e_1 + \vec{v}_1, \\ e_2 &= e_1 + \vec{v}_2, \\ e_3 &= e_1 + \vec{v}_1 + \vec{v}_2. \end{aligned}$$

Connect S to the four corners of the unit cell to get four triangles. At least one of these triangles has a right or obtuse angle. W.l.o.g., let the triangle be on points e_1, e_2, and S. Then both the vector from e_1 to S and the vector from e_2 to S is shorter than the vector from e_1 to e_2. Since at least one of the two is a quadrant II vector, we have a quadrant II vector shorter than $\vec{v}_2$. □

Our first main result is the following theorem. It establishes that if an array has basis vectors in both quadrants, then in a certain block of the array, which depends on the coefficients of the basis vectors, all symmetry vectors are linear combinations of the basis vectors. We state the theorem in terms of quadrant I for simplicity. Since the array can be rotated so that any quadrant becomes quadrant I, it applies to all quadrants.

THEOREM 3.9. *Let A be an array with basis vectors (r_1, c_1) and (r_2, c_2) in quadrants* I *and* II*, respectively, with $c_1 + c_2 < m$ and $r_1 + |r_2| < m$. Let L be an infinite lattice with the same basis vectors and containing the element $A[0, 0]$. Then, in the block $A[0 \ldots m - r_1 - |r_2|, 0 \ldots m - c_1 - c_2]$, an element is a quadrant* I *source iff it is a lattice element.*

Proof. By Lemma 3.6, if $S = A[r, c]$ is a lattice element, then it is a source. Suppose that S is not a lattice element, but that it is a quadrant I source. We will

show that S cannot occur within block $A[0 \ldots m - r_1 - |r_2|, 0 \ldots m - c_1 - c_2]$.

By way of contradiction, assume S does occur in prefix block $A[0 \ldots m - r_1 - |r_2|, 0 \ldots m - c_1 - c_2]$. There is a quadrant I vector $\vec{v}$ associated with S that is not a linear combination of $\vec{v}_1$ and $\vec{v}_2$. By Observation 3.7, copies of A can be aligned with the points of lattice L and the copies will be in register and cover the plane. Let $A' = A[r \ldots m-1, c \ldots m-1]$, i.e., the suffix block originating at element $A[r, c]$. Because S is a source, $\vec{v}$ maps $A[0 \ldots m - r - 1, 0 \ldots m - c - 1]$ to A'. For each copy of A, remove all but A'. The copies of A' are in register. Since A' has dimensions at least $r_1 + |r_2|$ by $c_1 + c_2$, it is at least as large as a unit cell of the lattice and therefore, the copies of A' also cover the plane. Now every element of the plane is mapped by $\vec{v}$ from an identical element, and there is a complete copy of A at S. S falls within some cell of lattice L. By Lemma 3.8, there is a quadrant I or quadrant II vector $\vec{v}_3$ from S to some corner e of the cell (or from e to S) which is *shorter* than the corresponding basis vector of L. Since there are complete copies of A at S and e, $\vec{v}_3$ is a symmetry vector and therefore, $\vec{v}_1$ and $\vec{v}_2$ are not both basis vectors of A as assumed. □

Since our quadrants are of size $\lceil \frac{m}{3} \rceil \times \lceil \frac{m}{3} \rceil$, they are no greater in size than the smallest block that can contain only lattice point sources. The region that contains only lattice point sources can be larger than the block described in Theorem 3.9 (see [GP-92]). Next, we prove the following important trait about radiant-periodic arrays that facilitates their handling in matching applications [AB-92b, **?**, KR-94]. Origins ($A[0, 0]$) of complete copies of a radiant-periodic array A that overlap without mismatch can be *ordered monotonically*.

DEFINITION 3.10. *A set of elements of an array B can be* ordered monotonically *if the elements can be ordered so that they have column index nondecreasing and row index nondecreasing (ordered monotonically in quadrant* I*) or row index nonincreasing (ordered monotonically in quadrant* II*).*

Our theorem is stated in terms of quadrant I, but generalizes to quadrant II.

THEOREM 3.11. *Let A be a radiant-periodic array with periodic vector in quadrant* I*. Let $S_1, \ldots, S_j$ be quadrant* I *sources occurring within quadrant* I*. On each source, place one copy of A by aligning $A[0, 0]$ with the source. If every pair of copies is in register, then the sources can be ordered monotonically in quadrant* I*.*

Proof. Suppose two sources $A[c_1, r_1]$ and $A[c_2, r_2]$ cannot be ordered monotonically. That is, $c_1 < c_2$ but $r_2 < r_1$. If there is no mismatch in the copies of A at these sources, then by the fact that $c_2 - c_1 < \frac{m}{3}$ and $r_1 - r_2 < \frac{m}{3}$, $\vec{v} = (r_2 - r_1, c_2 - c_1)$ is a periodic, quadrant II symmetry vector and by definition, A is lattice periodic, which is a contradiction. □

As stated earlier, our classification scheme applies to any rectangular array. The major modification is a new definition of length.

DEFINITION 3.12. *The length of a symmetry vector of a rectangular array is the maximum of the absolute values of its coefficients* scaled to the dimensions of the array*. Let A be n rows by m columns with $m \geq n$. Let $\vec{v} = (r, c)$ be a symmetry vector in A. Then the length of $\vec{v}$ scaled to the dimensions of the array is* $\max(r \cdot \frac{m}{n}, c)$.

4. Periodicity and witness algorithms. In this section, we present two algorithms, one serial and one parallel for finding all sources in an array A. In addition, for each location in A which is not a source, our algorithms find a *witness* that proves that the overlapping copies of A are not in register.

We want to fill out an array $Witness[-m - 1 \ldots m - 1, 0 \ldots m - 1]$. For each location $A[i, j]$ that is a quadrant I source, $Witness[i, j] = [m, m]$. Otherwise, $Witness[i, j] = [r, c]$ where $[r, c]$ identifies some mismatch. Specifically, $A[r, c] \neq$

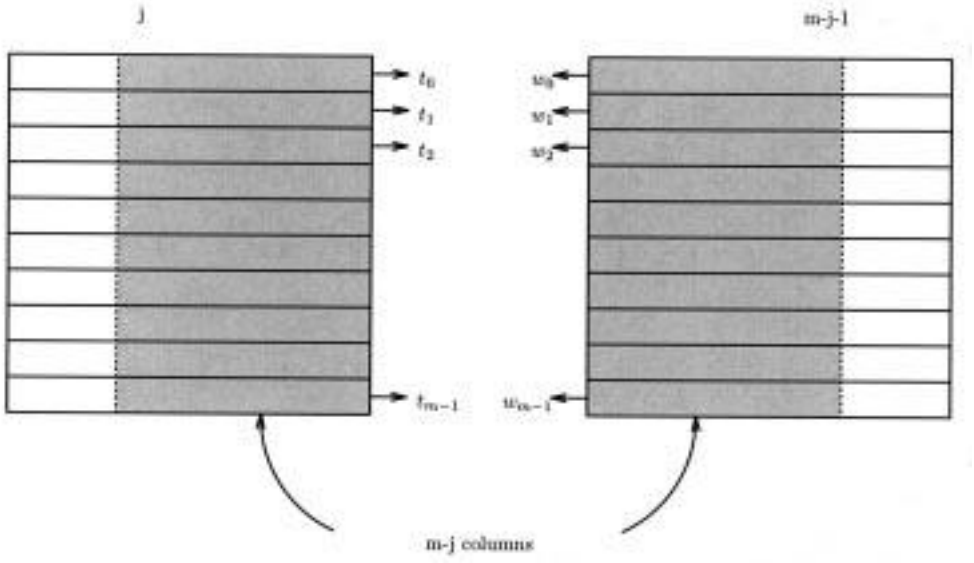


FIG. 10. *Representing a block of the array by a string.* $T_j = t_0 \dots t_{m-1}$ *is the text and* $W_j = w_0 \dots w_{m-1}$ *is the pattern.*

$A[i + r, j + c]$ (Figure 9). For each location $A[i, j]$ that is a quadrant II source, $Witness[i - (m - 1), j] = [m, m]$, otherwise $Witness[i - (m - 1), j] = [r, c]$, where $A[r, c] \neq A[i - (m - 1) + r, j + c]$.

4.1. The serial algorithm. Our serial algorithm (Algorithm A) makes use of two algorithms (Algorithms 1 and 2) from [ML-84] which are themselves variations of the KMP algorithm [KMP-77] for string matching. Algorithm 1 takes as input a pattern string w of length m and builds a table $lppattern[0 \dots m-1]$ where $lppattern[i]$ is the length of the longest prefix of w starting at w_i. Algorithm 2 takes as input a text string t of length n and the table produced by Algorithm 1 and produces a table $lptext[0 \dots n-1]$ where $lptext[i]$ is the length of the longest prefix of w starting at t_i.

The idea behind Algorithm A is the following: we convert the two-dimensional problem into a problem on strings (Figure 10). Let the array A be processed column by column and suppose we are processing column j. Assume we can convert the suffix block $A[0 \dots m-1, j \dots m-1]$ into a string $T_j = t_0 \dots t_{m-1}$ where t_i represents the suffix of row i starting in column j. This will serve as the text string. Assume also that we can convert the prefix block $A[0 \dots m-1, 0 \dots m-j-1]$ into a string $W_j = w_0 \dots w_{m-1}$, where w_i represents the prefix of row i of length $m - j$. This will serve as the pattern string. Now, use Algorithm 1 to produce the table *lppattern* for W_j and Algorithm 2 to produce the table *lptext* for T_j. If a copy of the pattern starting at t_i matches in every row to t_{m-1}, then $lptest[i] = m - i$ and $A[i, j]$ is a source. If the pattern doesn't match and the first pattern row to mismatch is row $k < m-i$, then $lptext[i] = k$ and $A[i, j]$ is not a source. The mismatch occurs between the prefix of pattern row k and the suffix of text row $i + k$. We need merely locate the mismatch to obtain the witness.

In order to treat the suffix and prefix of a row as a single character, we will build a *suffix tree* for the array. A suffix tree is a compacted trie of suffixes of a string $S = s_1 \cdots s_n$ [W-73]. Each node v has associated with it the indices [a,b] of some

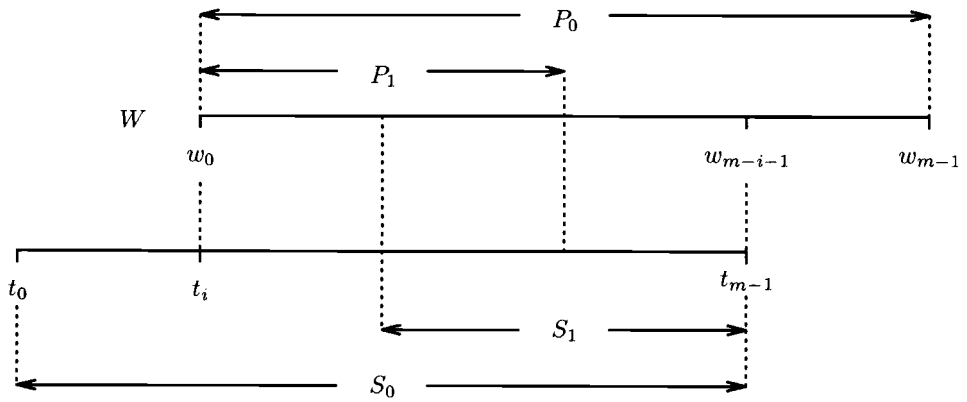


FIG. 11. *P_1 is a prefix of $t_i \dots t_{m-1}$ and S_1 is a suffix of $w_0 \dots w_{m-i-1}$.*

substring $S(v) = s_a \cdots s_b$ of S. If u is the least common ancestor (LCA) of two nodes v and w, then $S(u)$ is the longest common prefix of $S(v)$ and $S(w)$ [LV-85]. A tree can be preprocessed in linear time to answer LCA queries in constant time [HT-84]. Thus, we can answer questions about the length of $S(u)$ in constant time.

ALGORITHM A. *Serial algorithm for building a witness array and deciding periodicity class.*

Step A.1: Build a suffix tree by concatenating the rows of the array. Preprocess the suffix tree for least common ancestor queries in order to answer questions about the length of the common prefix of any two suffixes.

Step A.2: For each column j, fill out $Witness[0 \dots m-1, j]$ (quadrant I):

Step A.2.1: Use Algorithm 1 to construct the table *lppattern* for $W_j = w_0 \dots w_{m-1}$. Character w_i is the *prefix* of row i of length $m - j$. We can answer questions about the equality of two characters by consulting the suffix tree. If the common prefix of the two characters has length at least $m - j$ then the characters are equal.

Step A.2.2: Use Algorithm 2 to construct table *lptext* for $T_j = t_0 \dots t_{m-1}$. Character t_i is the *suffix* of row i starting in column j (also of length $m-j$). Again we test for equality by reference to the suffix tree.

Step A.2.3: For each row i, if $lptext[i] = m - i$ then we have found a quadrant I source and $Witness[i, j] = [m, m]$; otherwise, using the suffix tree, compare the suffix of text row $i + lptext[i]$ starting in column j with the prefix of pattern row $lptext[i]$. The length l of the common prefix will be less than $m - j$, and $Witness[i, j] = [lptext[i], l + 1]$.

Step A.3: Repeat Step A.2 for $Witness[-m+1 \dots 0, j]$ (quadrant II) by building the automata and processing the columns from the bottom up.

Step A.4: Select quadrant I and quadrant II basis vectors from $Witness$ if they exist.

Step A.5: Use the basis vectors to decide to which of four periodicity classes the pattern belongs.

THEOREM 4.1. *Algorithm A is correct and runs in time $O(m^2 \log |\Sigma|)$.*

Proof. The correctness of Algorithm A follows from the correctness of Algorithms 1 and 2 of [ML-84]. The suffix tree construction [W-73] takes time $O(m^2 \log |\Sigma|)$ while the preprocessing for least common ancestor queries [HT-84] can be done in

time linear in the size of the tree. Queries to the suffix tree are processed in constant time. The tables *lppattern* and *lptext* can be constructed in time $O(m)$ [ML-84]. For each of m columns, we construct two tables so the total time for Steps A.2 and A.3 is $O(m^2)$. Step A.4 can be done in one scan through the witness array and Step A.5 requires comparing all vectors to the basis vectors in order to distinguish between the radiant- and line-periodic classes, so the time for Steps A.4 and A.5 is $O(m^2)$. The total complexity of the pattern preprocessing is therefore $O(m^2 \log |\Sigma|)$. □

Recently, [GP-92] gave an $O(m^2)$ (linear time) serial algorithm for the witness computation.

4.2. The parallel algorithm. Our parallel algorithm (Algorithm B) makes use of the parallel string matching algorithm (Algorithm 3) from [V-85]. Algorithm 3 takes as input a pattern string w of length m and a text string t of length n and produces a boolean table $match[0 \ldots n-m-1]$, where $match[i] = true$ if a *complete* copy of the pattern starts at t_i. Algorithm 3 first preprocesses the pattern and then processes the text.

First, for a text of length m, we show how to modify Algorithm 3 to compute $match[0 \ldots m-1]$, where $match[i] = true$ if $t_i \ldots t_{m-1}$ is a prefix of the pattern. For simplicity, we assume m is a power of 2.

Let

$$\begin{aligned} P_k &= w_0 \ldots w_{\lfloor \frac{m-1}{2^k} \rfloor}, \\ S_k &= t_{m-1-\lfloor \frac{m-1}{2^k} \rfloor} \ldots t_{m-1}, \\ k &= 0, 1, \ldots, \log m. \end{aligned}$$

For example, P_1 is the prefix of w of length $\frac{m}{2}$ and S_1 is a suffix of t of the same length. The following observation embodies the key idea (Figure 11).

OBSERVATION 4.2. *If $t_i \ldots t_{m-1}$ is a prefix of w of length between m and $\frac{m}{2}$, then P_1 is a prefix of $t_i \ldots t_{m-1}$ and S_1 is a suffix of $w_0 \ldots w_{m-i-1}$. Similarly, if $t_i \ldots t_{m-1}$ is a prefix of w of length between $\frac{m}{2}$ and $\frac{m}{4}$, then the prefix and suffix are P_2 and S_2, etc.*

Now, for each $k \geq 1$, we attempt to match P_k in S_{k-1} and S_k in P_{k-1}. If a matched prefix begins at t_i and a matched suffix ends at w_{m-i-1} then $t_i \ldots t_{m-1}$ is a prefix of w.

Using Algorithm 3, we first preprocess the P_k and S_k as patterns and then use these to process the appropriate segments as text. We can additionally modify Algorithm 3 so that at every index where a prefix or suffix does not match, we obtain the location of a mismatch. Since the sum of the lengths of the P_i and S_i are no more than a linear multiple of the length of w, the modification does not increase the complexity of the algorithm and therefore the time complexity of the modified Algorithm 3 is $O(\log m)$ using $O(\frac{m}{\log m})$ CRCW processors—the same as the unmodified algorithm [V-85]. In our parallel algorithm, only Step 2 differs from the serial algorithm.

ALGORITHM B. *Parallel algorithm for finding sources and building a witness array.*

Step B.2: For each column j, fill out $Witness[0 \ldots m-1, j]$ (quadrant I):
  Step B.2.1: For each $k = 1, \ldots, \log m$:
    Step B.2.1.1: Use W_j to form P_k and P_{k-1} and T_j to form S_k and S_{k-1}. Use modified Algorithm 3 to match P_k in S_{k-1} and S_k in P_{k-1}. As in the serial algorithm, use the suffix tree to answer questions about equality.

Step B.2.1.2: For each row i for $m-1-\frac{m-1}{2^{k-1}} \leq i < m-1-\frac{m-1}{2^k}$. If P_k matches beginning at t_i and S_k matches ending at w_{m-i-1}, then $Witness[i,j] = [m,m]$. Otherwise, using the row r of mismatch from modified Algorithm 3, refer to the suffix tree to find the column c of mismatch and set $Witness[i,j] = [r,c]$.

THEOREM 4.3. *Algorithm B is correct and runs in time $O(\log m)$ using $O(m^2)$ CRCW processors.*

Proof. The suffix tree construction [AILSV-87] and preprocessing for LCA queries [SV-88] are done in time $O(\log m)$ using $O(m^2)$ CRCW processors. Step B.2 is done in time $O(\log m)$ using $O(\frac{m^2}{\log m})$ CRCW processors [V-85]. Finding the basis vectors is done by prefix minimum [LF-80] in time $O(\log m)$ using $O(\frac{m^2}{\log m})$ processors. Distinguishing the line and radiant periodic cases can be done in constant time using $O(m^2)$ processors. The total complexity is therefore $O(\log m)$ time using $O(m^2)$ CRCW processors. □

REFERENCES

[AB-92a] A. AMIR AND G. BENSON, *Two-dimensional periodicity and its application*, in 3rd ACM-SIAM Symposium on Discrete Algorithms, Orlando, FL, SIAM, Philadelphia, 1992, pp. 440–452.

[AB-92b] A. AMIR AND G. BENSON, *Efficient two-dimensional compressed matching*, in Proc. Data Compression Conference, Snowbird, Utah, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 279–288.

[ABF-93] A. AMIR, G. BENSON, AND M. FARACH, *Optimal parallel two dimensional text searching on a CREW PRAM*, in Proc. 5th Annual Symposium on Parallel Algorithms and Architectures, ACM, New York, 1993, pp. 79–85.

[ABF-94] A. AMIR, G. BENSON, AND M. FARACH, *An alphabet independent approach to two-dimensional matching*, SIAM J. Comput., 23 (1994), pp. 313–323.

[ABF-97] A. AMIR, G. BENSON, AND M. FARACH, *Optimal two dimensional compressed matching*, J. Algorithms, 24 (1997), pp. 354–379.

[AC-75] A. V. AHO AND M. J. CORASICK, *Efficient string matching*, Comm. ACM, 18 (1975), pp. 333–340.

[AF-91] A. AMIR AND M. FARACH, *Efficient 2-dimensional approximate matching of non-rectangular figures*, in Proc. 1st ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, SIAM, Philadelphia, 1990, pp. 212–223.

[AL-91] A. AMIR AND G. M. LANDAU, *Fast parallel and serial multidimensional approximate array matching*, Theoret. Comput. Sci., 81 (1991), pp. 97–115.

[ALV-90] A. AMIR, G. M. LANDAU, AND U. VISHKIN, *Efficient pattern matching with scaling*, in Proc. 1st ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, SIAM, Philadelphia, 1990, pp. 344–357.

[AILSV-87] A. APOSTOLICO, C. ILIOPOULOS, G. M. LANDAU, B. SCHIEBER, AND U. VISHKIN, *Parallel construction of a suffix tree with applications*, Algorithmica, 3 (1988), pp. 347–365.

[B-78] T. P. BAKER, *A technique for extending rapid exact-match string matching to arrays of more than one dimension*, SIAM J. Comput., 7 (1978), pp. 533–541.

[Bi-77] R. S. BIRD, *Two-dimensional pattern matching*, Inform. Process. Lett., 6 (1977), pp. 168–170.

[BM-77] R. S. BOYER AND J. S. MOORE, *A fast string searching algorithm*, Comm. ACM, 20 (1977), pp. 762–772.

[CCG+93] R. COLE, M. CROCHEMORE, Z. GALIL, L. GĄSIENIEC, R. HARIHAN, S. MUTHUKRISHNAN, AND K. PARK, *Optimally fast parallel algorithms for preprocessing and pattern matching in one and two dimensions*, in Proc. 34th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1993, pp. 248–258.

[CR-92] M. CROCHEMORE AND W. RYTTER, *On two-dimensional pattern matching by optimal parallel algorithms*, Theoret. Comput. Sci., 132 (1994), pp. 403–414.

[G-85] Z. Galil, *Optimal parallel algorithms for string matching*, Inform. and Control, 67 (1985), pp. 144–157.

[GP-92] Z. Galil and K. Park, *Truly alphabet independent two-dimensional pattern matching*, in Proc. 33rd IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 247–256.

[HT-84] D. Harel and R. E. Tarjan, *Fast algorithms for finding nearest common ancestors*, SIAM J. Comput., 13 (1984), pp. 338–355.

[KR-94] M. Karpinski and W. Rytter, *Alphabet independent optimal parallel search for 3-dimensional patterns*, in Proc. 5th Annual Symposium on Combinatorial Pattern Matching, Lecture Notes in Comput. Sci. 807, Springer-Verlag, Berlin, 1994, pp. 125–135.

[KMP-77] D. E. Knuth, J. H. Morris, and V. R. Pratt, *Fast pattern matching in strings*, SIAM J. Comput., 6 (1977), pp. 323–350.

[KS-87] K. Krithivasan and R. Sitalakshmi, *Efficient two-dimensional pattern matching in the presence of errors*, Inform. Sci., 47 (1987), pp. 169–184.

[LF-80] R. E. Ladner and M. J. Fischer, *Parallel prefix computation*, J. ACM, 27 (1980), pp. 831–838.

[LV-85] G. M. Landau and U. Vishkin, *Efficient string matching in the presence of errors*, in Proc. 26th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1985, pp. 126–136.

[ML-84] M. G. Main and R. J. Lorentz, *An $O(n \log n)$ algorithm for finding all repetitions in a string*, J. Algorithms, 5 (1984), pp. 422–432.

[RR-93] M. Regnier and L. Rostami, *A unifying look at d-dimensional periodicities and space coverings*, in Proc. 4th Symposium on Combinatorial Pattern Matching, Lecture Notes in Comput. Sci. 684, Springer-Verlag, New York, 1993, pp. 215–227.

[SV-88] B. Schieber and U. Vishkin, *On finding lowest common ancestors: Simplification and parallelization*, SIAM J. Comput., 17 (1988), pp. 1253–1262.

[V-85] U. Vishkin, *Optimal parallel pattern matching in strings*, Inform. and Control, 67 (1985), pp. 91–113.

[V-91] U. Vishkin, *Deterministic sampling—a new technique for fast pattern matching*, SIAM J. Comput., 20 (1991), pp. 303–314.

[W-73] P. Weiner, *Linear pattern matching algorithms*, in Proc. 14th IEEE Symposium on Switching and Automata Theory, IEEE Computer Society Press, Los Alamitos, CA, 1973, pp. 1–11.

SIAM J. COMPUT.
Vol. 27, No. 1, pp. 107–119, February 1998
© 1998 Society for Industrial and Applied Mathematics
005

A FAST DISCRETE APPROXIMATION ALGORITHM FOR THE RADON TRANSFORM*

MARTIN L. BRADY†

Abstract. This paper addresses fast parallel methods for the computation of the Radon (or Hough) transform. The Radon transform of an image is a set of projections of the image taken at different angles. Its computation is important in image processing and computer vision for problems such as pattern recognition and reconstruction of medical images. A unique new method for combining partial results is presented, from which an algorithm is constructed that computes a provably good approximation to the discrete Radon transform. The approximate discrete Radon transform (ADRT) algorithm computes $4N-4$ projections through an $N \times N$ image in time $O(N^2 \lg N)$ (the majority of previous algorithms are $O(N^3)$). The method is quite simple and easy to parallelize. A parallel version of the ADRT requires only $O(\lg N)$ parallel steps on $O(N^2)$ processors, ignoring communication time. An additional property of the algorithm is that it can be applied directly to compute the backprojection step of the inverse RT.

Key words. Radon transform, Hough transform, approximation algorithms, image processing, image reconstruction

AMS subject classifications. 68Q20, 68U10

PII. S0097539793256673

1. Introduction.

The problem of computing projections of two-dimensional data arises in numerous applications in image processing and computer graphics. The Radon transform (RT) of an image is a set of projections of the image along lines taken at different angles (i.e., a mapping from image space to projection space, or "Radon space"). For discrete image data, a projection is computed by summation of all data points that lie within specified unit-width strips. The Hough transform is a restricted form of the discrete RT (DRT) in which the input is taken to be discrete binary data [9]. The Hough transform is an essential component of many methods for pattern recognition and parameter extraction [12], and it has been applied to problems such as line and curve detection [5], [1].

The standard computation of the RT of an $N \times N$ image at M different angles requires time $O(N^2 M)$. In this paper, we present a new approach to the problem and develop an algorithm that computes a good *approximation* to the DRT. This algorithm computes the RT for a specific set of $4N-4$ projection angles in only $O(N^2 \lg N)$ time.[1] The algorithm is extremely simple, and the approximation is very good even for large images. The maximum sampling error grows only as $\lg N$ times a small constant (< 1).

The inversion of the RT is also of importance in image processing. It is essential in reconstructing images collected by projective techniques; for instance, it is used in computer tomography to reconstruct medical images from data collected with X rays (e.g., computed tomography (CT), or magnetic resonance imaging (MRI)); (see [11]). The inversion (reconstruction of an image from its projections) of the RT can be achieved by a fixed convolution, followed by backprojection (the inversion formula

*Received by the editors October 8, 1993; accepted for publication (in revised form) December 30, 1995. This work was supported in part by a grant from IBM under the Technical Interchange Program and by Lockheed Missiles and Space Co., Inc. internal research funds.
http://www.siam.org/journals/sicomp/27-1/25667.html
†Microcomputer Research Labs, Intel Corp., Santa Clara, CA 95052 (Martin_Brady@ccm.sc.intel.com).
[1]In this paper "lg" denotes a base 2 logarithm.

was discovered by J. Radon [15]). Backprojection is similar to the RT projection computation, except that the projections do not usually follow straight lines. However, our algorithm produces nonuniformly sampled *linogram* Radon data. A consequence is that data, sampled as in our forward RT algorithm, can be backprojected using the same algorithm. As a result, our algorithm for the DRT is also useful in computing the inverse DRT.

In the next section we give some definitions and describe previous work done in the area. Section 3 contains the description of our new algorithm. In section 4 we describe the application of the algorithm to the computation of the inverse of the RT. In section 5 we conclude with a summary of the results as well as observations about other problems to which our method can be applied.

2. Description of the RT. The two-dimensional RT of an image defined in (x, y) coordinate space is a set of projections of the image taken by integrating along the set of lines defined by $x\cos\theta + y\sin\theta = d$ for $0 \le \theta < \pi$. Parameterized in this way, θ is the angle of the line with respect to the positive y-axis and d represents its distance from the origin. An image in (x, y) space is thus transformed into "Radon space" (d, θ).

When working with digital images, a discretized form of the RT is required. A digital image $I(x, y)$ is an $N \times N$ array of *pixels* $(x, y) \in \mathbb{Z}^2$, $0 \le x, y \le N-1$, each representing the average gray level of a unit square in the image.[2] The gray levels can be taken to be nonnegative reals $I(x, y)$: $\mathbb{R}^2 \to \mathbb{R}_{>0}$ (although in practice gray levels are often restricted to eight bit integers, $0 \le I(x, y) \le 255$). A line integral along $x\cos\theta + y\sin\theta = d$ is approximated by a summation of the pixels lying in the one-pixel-wide *strip* $d - \frac{1}{2} \le x\cos\theta + y\sin\theta < d + \frac{1}{2}$ (see Figure 1). Since strips have unit width, d can be restricted to integer values, and for a given θ at most $\sqrt{2}N$ strips are needed. The number of angles is defined to be M, and we assume here that $M = \Theta(N)$. The set of M angles is often defined to be uniformly distributed between 0 and π (however, we will explore the benefits of a nonuniform distribution in this paper).

The DRT can be computed as follows. For any given angle θ, each of the pixels lies in exactly one strip. Therefore, for each pixel we simply determine the strip to which it belongs (d relative to θ) and add the pixel's value to the current total for strip (d, θ). This procedure is repeated for each value of θ. A simple pseudocode description of this algorithm is given in Algorithm 1. The complexity of this method is $O(N^2M)(= O(N^3)$ under the assumption that $M = \Theta(N))$.

ALGORITHM 1. *Standard DRT calculation.*

Initialization:
$R(d, \theta) = 0$ for all (d, θ);

DRT computation:

for $\theta = 0$ **to** $\frac{(M-1)\pi}{M}$ **step** $\frac{\pi}{M}$
{ **for** $x = 0$ **to** $N-1$
{ **for** $y = 0$ **to** $N-1$

[2] Actually, the image can be more generally defined as an $N_1 \times N_2$ array of pixels, where N_1 is not necessarily equal to N_2. However, the discussion is greatly simplified by limiting it to square arrays. The extension to nonsquare images could most easily be handled by zero-padding the image in the smaller dimension.

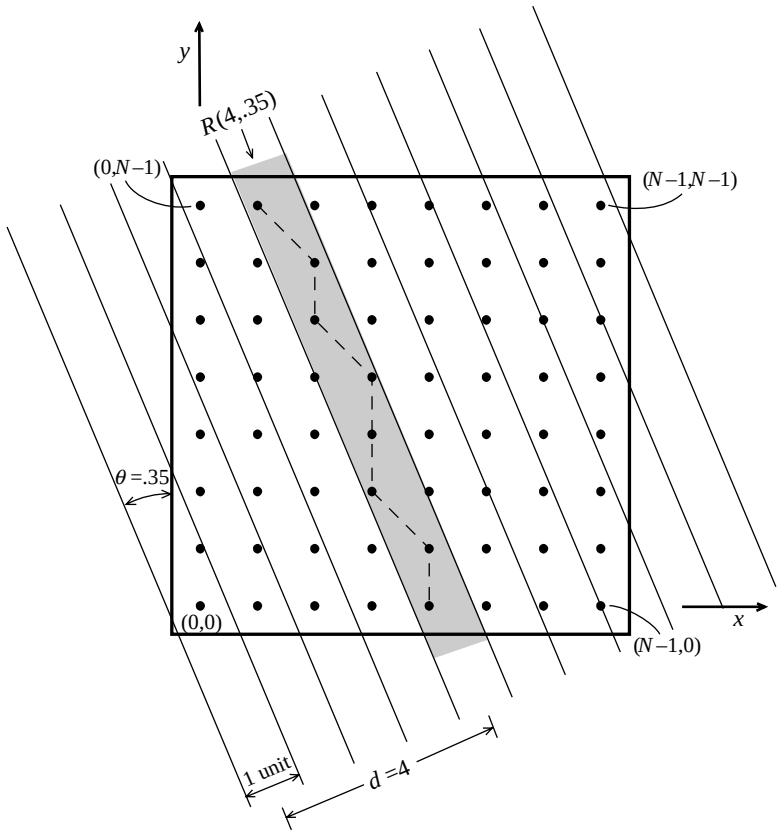
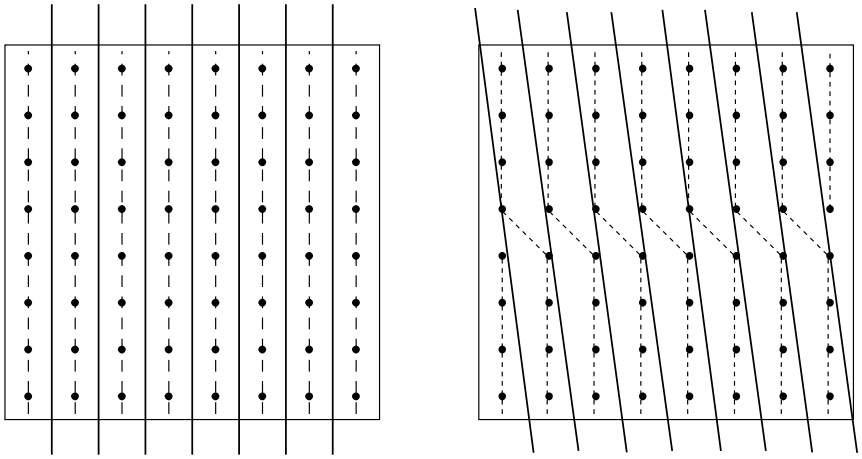


FIG. 1. *Representation of the set of strips for summation along a single direction, θ. Each strip represents a single Radon data point (the sum of the pixels in the shaded strip represents Radon data point $(d, \theta) = (4, .35)$).*

```
            {     d = ⌊x cos θ + y sin θ + 1/2⌋;
                  R(d, θ) = R(d, θ) + I(x, y);
            }
      }
}
```

Due to the complexity of computing the RT and its importance in image processing and other fields, much research has been devoted to speeding it up by mapping the problem into parallel processing architectures. Implementations that require $O(N)$ time on an $N \times N$ mesh-connected array of processors were discovered independently by Cypher, Sanz, and Snyder [4] and Guerra and Hambrush [8]. Pan and Chuang [14] give an $O(\lg N)$ time algorithm for a hypercube with $O(N^4)$ processors, and Jenq and Sahni [10] presented a collection of algorithms for reconfigurable meshes. Many special-purpose parallel architectures have also been proposed; e.g., a pipeline architecture solution was given by Sanz and Hinkle [17].

Our new approach is to construct a simple sequential *approximation* algorithm that is both asymptotically and practically more efficient than the standard algorithm. It requires only $O(N^2 \lg N)$ time to compute projections along $4N - 4$ angles in an $N \times N$ image—a sequential speedup of $\Theta(N/\lg N)$. This large speedup is obtained at



(a) Angle θ_1. (b) Angle θ_2.

FIG. 2. *The large amount of overlap between strips at neighboring angles is illustrated. Dashed lines trace out the sets of points that compose individual strips. Notice that groups of four points are common to one strip at each of the two angles.*

the cost of computing only a fixed, nonuniformly distributed set of angles and computing only an approximation to the desired strip sums. We describe our approximate discrete Radon transform (ADRT) algorithm in section 3.

3. The ADRT algorithm. In Algorithm 1, each point (d, θ) in Radon space is computed independently by summing the pixels within each strip (d, θ). However, the computation can be sped up by observing that *for discrete images*, the computations for different points in the discrete Radon space are not entirely independent; specifically, observe that for neighboring angles, large subsets of pixels may be shared by different strips, as illustrated in Figure 2. Thus, one could potentially save time by computing such shared partial sums only once for use in two or more lines. One approach is to attempt to derive the optimal ordering of computations for a given problem size to make maximum use of partial terms. Unfortunately, there does not appear to be much structure to these sets of partial sums, and it may be more difficult to calculate the proper subsets and order the computations accordingly than to simply calculate the DRT data independently. Instead, we construct an algorithm to compute an *approximation* to the DRT which is designed to take maximum advantage of intermediate terms in a structured manner. We define a specific set of angles and distances and define a set of approximate discrete lines which make full use of partial sums. The number of sample points (d, θ) that we compute is sufficiently dense so that arbitrary data points in Radon space can be extracted by interpolation, if desired. (The accuracy of this approximation is discussed in section 3.3.)

In order to simplify the discussion, let N be a power of two. The extension to arbitrary N can be achieved by either zero-padding the data or by complicating the algorithm's handling of image boundaries, without increasing the asymptotic run time.

3.1. General strategy. We first describe the general strategy of our algorithm. Partial sums of the gray levels of consecutive pixels, representing short discrete line segments, will be computed. We refer to these sets of pixels as *segments* and to the

sum of the gray levels of the pixels in a segment as the *value* of the segment. In the first pass, the values of a large set of two-point segments are computed. Next, pairs of two-point segments are combined to form a set of four-point segments. In successive passes, the values of 2^i-point segments are computed using only the values of the 2^{i-1}-point segments from the previous pass. After $\lg N$ passes, the values of N-point segments have been computed, each representing the sum of a strip of N pixels from the original image. These sums constitute the approximate DRT data.

The computation is divided into four parts which correspond to four equal-sized ranges of angles $\frac{\pi}{4}r \le \theta \le \frac{\pi}{4}(r+1)$ for $r = 0, 1, 2, 3$. Within a range of 45 degrees, the angles are similar enough to derive benefit from sharing intermediate terms. The algorithm will be described for a single group of angles $0 \le \theta \le \pi/4$; each of the other three ranges can be computed symmetrically.

Central to our method is the specification of a set of segments which are simply defined yet closely approximate straight lines. Let the two end points of a segment be labeled (x_1, y_1) and (x_2, y_2). The *x-displacement* of the segment is defined as $|x_1 - x_2|$ (the *y-displacement* is defined analogously). First, we specify the set of segments computed in each pass i by their end points. We then explain how the segments in pass $i+1$ are approximated using the pass i segments. The segments computed as pass i have the following properties.

1. Both end points of each segment lie on pixels in the image space (i.e., the end points have integral (x, y) values).

2. The value of each segment computed in pass i represents the sum of 2^i pixels. One pixel is taken from each of 2^i consecutive rows. Thus, the y-displacement of each segment is $2^i - 1$ and the x-displacement determines the segment's angle.

3. The number of different angles represented by the segments computed in pass i is 2^i. As noted above, a segment's angle depends solely on its x-displacement. Since the angles are between 0 and $\pi/4$, the x-displacements are between 0 and $2^i - 1$. All end points are integer, and therefore the set of x-displacements at pass i is $\{0, 1, 2, \ldots, 2^i - 1\}$. The set of angles for pass i can be calculated as follows:

$$\theta = \tan^{-1}\left(\frac{a}{2^i - 1}\right) \quad \text{for } a = 0, 1, 2, \ldots, 2^i - 1. \tag{1}$$

4. In the y-direction, segments start only on every 2^ith row, i.e., rows y such that $y \equiv 0 \pmod{2^i}$. Within this set of rows, all pixels are valid starting points. Thus, the lowest set of segments spans rows 0 through $2^i - 1$, the next set spans 2^i through $2^{i+1} - 1$, etc. More specifically, the set of segments computed in pass i begins at $y = j(2^i)$ and ends at $y = (j+1)(2^i) - 1$, for $j = 0, 1, 2, \ldots, (N/2^i) - 1$, and spans all x-values. The segments constructed in the first three passes of the algorithm are illustrated for an 8×8 portion of an image in Figure 3.

The properties given above describe the situation for strips that have their lower and upper end points at the bottom and top rows of the image, respectively. However, some strips "fall off" the left or right boundaries of the image and therefore do not sum an entire set of N pixels. A simple way to handle these boundary conditions is to pad the image with zeros and construct the segments in cyclic fashion. Specifically, the image is padded with zeros in the range $N \le x < 2N$, $0 \le y < N$ into an $N \times 2N$ image, and all x-values are taken modulo $2N$. Strips that "fall off" of the left side of the image wrap around to sum zeros from the padded right half, and some of the strips that begin in the zero-padded right half of the image eventually enter the left half and complete the sum for a line that falls off of the right half. As a result, all of

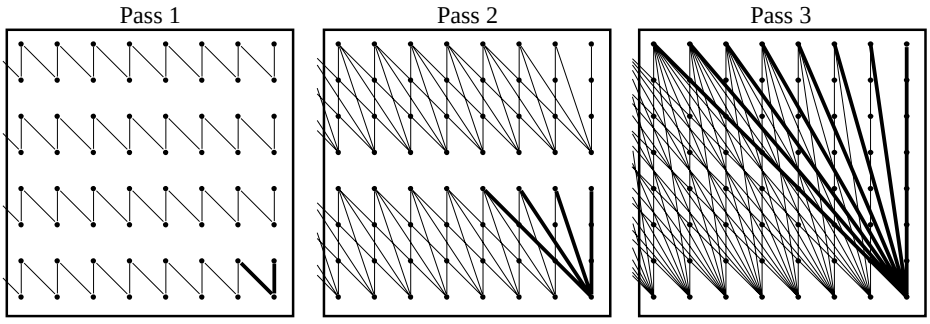


FIG. 3. *Illustration of the segments computed in the first three passes of the ADRT algorithm.*

the strips in the cyclically defined image consist of exactly N pixels. Strips that fall off of an edge take the balance of their N pixels from the set of padded zeros. Since the maximum x-displacement is $N-1$, a strip that moves into the padded zeros never passes all the way through to reenter the image from the other side.

3.2. Algorithm specification. In pass i, two segments of the same angle, $\tan^{-1}(\frac{\lfloor a/2 \rfloor}{2^{i-1}-1})$, are adjoined to create a segment that is about twice as long whose angle is $\tan^{-1}(\frac{a}{2^i-1})$. Let $R_i(x,y,a)$ represent the value of the segment constructed in pass i with lower end point at (x,y) and x-displacement a. Then $R_i(x,y,a)$ is computed as the sum of the values of two (2^{i-1})-point segments from pass $i-1$ as follows (recall that x is taken modulo $2N$):

$$R_i(x,y,a) = R_{i-1}(x,y,\lfloor a/2 \rfloor) + R_{i-1}(x-\lceil a/2 \rceil, y+2^{i-1}, \lfloor a/2 \rfloor) \tag{2}$$

for $x = 0,1,2,\ldots,2N-1$, for $y = j(2^i)$ such that $j = 0,1,2,\ldots,(N/2^i)-1$, and for $a = 0,1,2,\ldots,2^i-1$. Note that the ranges of both y and a are dependent on i. Their product is exactly N, for all i, and therefore within each pass a total of exactly $2N^2$ data values are maintained. At each consecutive pass, the number of different starting point y-values is halved, but the number of different angles per starting point doubles.

The ADRT algorithm is summarized below in Algorithm 2. The image $I(x,y)$ is represented as $R_0(x,y,0)$. After $\lg N$ passes, the DRT data $R(d,\theta)$ are obtained from $R_{\lg N}(x,0,a)$, i.e., $R(d,\theta) = R_{\lg N}(x,0,a)$ for $\theta = \tan^{-1}(\frac{a}{N-1})$ and $d = x\cos\theta$.

ALGORITHM 2. *ADRT computation for* $0 \le \theta \le \pi/4$.

Initialize R_0:

$R_0(x,y,0) = I(x,y)$ for $0 \le x,y < N$
$R_0(x,y,0) = 0$ for $N \le x < 2N, 0 \le y < N$

Approximate DRT computation:

```
for i = 1 to lg N
{    for a = 0 to 2^i - 1
     {    for y = 0 to N - 2^i step 2^i
          {    for x = 0 to 2N - 1
               {    R_i(x,y,a) = R_{i-1}(x,y,⌊a/2⌋) + R_{i-1}(x - ⌈a/2⌉, y + 2^{i-1}, ⌊a/2⌋)
               }
          }
     }
}
```

A total of $O(N^2 \lg N)$ simple arithmetic calculations dominate the computation time in Algorithm 2. Three more applications of symmetric variations of Algorithm 2 are performed to calculate the DRT data for angles in the range $\pi/4$ through π. The basic computation at a single step is extremely simple, requiring only a few additions and divisions by 2. All of the $2N^2$ segment sums in a given pass i can be performed independently, so a PRAM implementation on $2N^2$ processors would require $O(\lg N)$ time. Hence, this algorithm is both asymptotically and practically fast and should yield efficient sequential and parallel implementations. (References [16] and [13] describe sequential and parallel ADRT implementations, respectively; mappings into various parallel architectures are discussed in [3].)

3.3. Analysis of the ADRT. By their construction, the segments computed by Algorithm 2 are "crooked" approximations of the desired strips. Each pass of Algorithm 2 introduces some deviation, and these errors may be compounded in each pass. In this section, we derive an upper bound on the deviation of a pixel that is used to compute $R(d, \theta)$ from the line (d, θ). (Recall that in the standard DRT specified in Algorithm 1, all pixels used to compute a line (d, θ) are within $\frac{1}{2}$ unit of the line.)

Consider the construction of a *parent* segment S in pass i by adjoining two *child* segments S_1 and S_2. Assume for now that S_1 and S_2 are exactly represented, i.e., that all samples lie on the child segments. Observe that each child segment intersects the parent at one of its end points and reaches its maximum distance from S near the parent's center (see Figure 4). We refer to the maximum separation between two line segments taken in the x-direction as their *horizontal separation*.

LEMMA 1. *The horizontal separation of a segment S constructed by Algorithm 2 from either of the two subsegments S_1 and S_2 used to construct it is less than $\frac{1}{2}$ unit.*

Proof. Consider w.l.o.g. a segment S with a lower end point at $(0, 0)$ and x-displacement a, constructed in pass i. Its children S_1 and S_2 are adjoined either diagonally or vertically, depending upon whether a is odd or even, as shown in Figure 4.

Case 1. *a odd (diagonally joined).* This case is illustrated in Figure 4a. The coordinates of the upper end point of S_1 are $(-(a-1)/2, 2^{i-1}-1)$. Since the slope of S is $-\frac{2^i-1}{a}$, at $y = 2^{i-1}-1$ segment S has x-coordinate $x = \frac{-a(2^{i-1}-1)}{2^i-1}$. The horizontal separation of S_1 and S is therefore

$$-\frac{a-1}{2} - \frac{-a(2^{i-1}-1)}{2^i-1} = -\frac{a}{2} + \frac{1}{2} + a\left(\frac{2^{i-1}-1}{2^i-1}\right) < -\frac{a}{2} + \frac{1}{2} + a\frac{1}{2} = \frac{1}{2}.$$

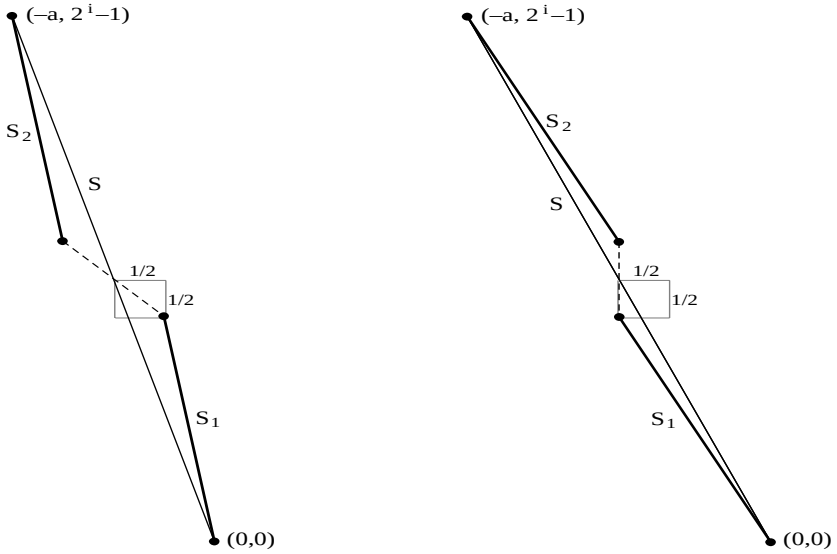
Since S_1 and S_2 are symmetric, the horizontal separation of S_2 and S is also less than $1/2$.

Case 2. *a even (vertically joined).* This case is illustrated in Figure 4b. Here the upper end point of S_1 is at $(-a/2, 2^{i-1}-1)$ and the x-coordinate of S at $y = 2^{i-1}-1$ is again $x = \frac{-a(2^{i-1}-1)}{2^i-1}$. The horizontal separation of S_1 and S is

$$\frac{-a(2^{i-1}-1)}{2^i-1} - \left(-\frac{a}{2}\right) = -\frac{a}{2(2^i-1)}.$$

At pass i, $a \le 2^i - 1$ in general, and since a is even, $a \ne 2^i - 1$, so the horizontal separation is strictly less than $\frac{2^i-1}{2(2^i-1)} = \frac{1}{2}$. (Again S_1 and S_2 are symmetric, so the bound holds for S_2 as well.) □

Lemma 1 bounds the error between a parent segment and two exactly represented child segments used to construct it. But in the ADRT, the children are approximations



(a) S_1 and S_2 adjoined diagonally when a is odd.

(b) S_1 and S_2 adjoined vertically when a is even.

FIG. 4. *The values of two 2^{i-1}-point segments S_1 and S_2 of x-displacement $\lfloor a/2 \rfloor$ are added to create a 2^i-point segment of x-displacement a starting at $(0,0)$. When a is odd, S_2 starts one unit above and one unit to the left of the upper end point of S_2; if a is even, S_2 starts just one unit above the upper end point of S_2.*

themselves and contain some error in their constructions and so forth. The maximum total horizontal error (the horizontal distance of the N-point line segment S from the *points* used to construct it) can be bounded by repeated application of Lemma 1. In the first pass, simple two-point segments are created, incurring no error. Next, these segments are combined to form four-point segments, whose points lie within $\frac{1}{2}$ unit in the x-direction from the intended line. In the next pass, these four-point segments are combined to form eight-point segments, and an additional error of less than $\frac{1}{2}$ unit in the x-direction is incurred. That is, the maximum horizontal separation of the eight-point segments from the (exactly represented) child four-point segments is at most $\frac{1}{2}$, and the separation of these four-point segments from the two-point segments used to construct them is an additional $\frac{1}{2}$. The $\frac{1}{2}$-unit horizontal error bound is invoked for each pass of the algorithm from 2 to $\lg N$. Some of the errors may offset each other, but in the worst case all horizontal errors could be incurred in the same direction, so the maximum total horizontal distance of a pixel from one of the lines to which it contributes is $\frac{1}{2}$ $(\lg\ N - 1)$. The total perpendicular distance is therefore $\frac{1}{2}$ $(\lg\ N - 1)\cos\theta \leq \frac{1}{2}(\lg\ N - 1)$.

The following theorem summarizes the performance of Algorithm 2.

THEOREM 1. *Algorithm 2 produces a set of approximate DRT data points (d, θ) for $\theta = \tan^{-1}(a/(N-1))$, $a = 0, 1, 2, \ldots, N-1$, and $d = d_x \cos\theta$, $d_x = 0, 1, 2, \ldots, 2N-1$, such that all pixels used to generate (d, θ) are within $\frac{1}{2}(\lg N - 1)\cos\theta$ units of the line parameterized by (d, θ). The algorithm requires $O(N^2 \lg N)$ time on a sequential machine, and $O(\lg N)$ parallel steps using $O(N^2)$ processors on an EREW PRAM.*

Note that the error bound is somewhat pessimistic; no single error actually reaches

TABLE 1
Calculated maximum horizontal and perpendicular sampling errors in the ADRT.

N	Maximum horizontal error	Maximum perpendicular error
4	0.333	0.316
8	0.429	0.424
16	0.667	0.632
32	0.774	0.764
64	1.000	0.949
128	1.110	1.095
256	1.333	1.265
512	1.444	1.425
1024	1.667	1.581
2048	1.778	1.754
4096	2.000	1.897

$\frac{1}{2}$ unit, and it is not possible for a single point to receive the maximum error at every iteration. Numerical calculations suggest that the maximum horizontal error may in fact be bounded by $\frac{\lg N}{6}$ (see Table 1).

Notice that the Radon data produced by Algorithm 2 are not uniformly distributed in either d or θ. However, the $2N^2$ data points are extremely dense for an $N \times N$ image. Spacing between adjacent lines ranges from one pixel (for $\theta = 0$) to $1/\sqrt{2}$ pixels (for $\theta = \pi/4$). These lines are as densely spaced as the individual pixels; there is little to be gained by increasing their density without resorting to interpolation. Furthermore, the line spacings are optimized to the geometry of the data points, in that line segments start and end centered on pixels and become more dense at angles in which pixels appear more densely distributed. Similarly, the angles are not uniformly distributed but are dense. For example, the consecutive angles 0 and $\tan^{-1}(\frac{1}{N-1})$ (i.e., $a = 0$ and $a = 1$) with the same starting point share 50% of their data points. It is therefore unlikely that the Radon data computed by Algorithm 2 is too sparse.

It may be the case that specific data points are desired. In this case, it would be reasonable to calculate the desired points from the Algorithm 2 data by interpolation, or to simply select the nearest ADRT data points. There is always an ADRT line within $\frac{1}{2}$ unit of horizontal separation of the desired line. Consider the range $0 \le \theta \le \pi/4$, for example. Since all integer x-coordinates on $y = 0$ and $y = N - 1$ are valid starting and ending points for ADRT lines, one can always select an ADRT sample whose end points are both within $\frac{1}{2}$ unit in the x-direction of the desired line. The maximum horizontal separation will be at one of the end points and is thus bounded by $\frac{1}{2}$. Thus, the ADRT data can be used to approximate *any* Radon data point with a maximum perpendicular error of $\frac{1}{2} \lg N$ units.

3.4. Reducing the approximation error. The maximum error in the data points used to compute the ADRT grows very slowly with the size of the image. This fact can be used to further reduce the error with only a small factor increase in computation time. The general method is as follows. First, the image is *expanded* by a factor k in both dimensions (k must be a power of 2), and then the ADRT is applied. The maximum error incurred for the $kN \times kN$ image is $\frac{1}{2}(\lg kN)$—only $\frac{1}{2}(1 + \lg k)$ units greater than that of an $N \times N$ image. When the RT data is *contracted* back to its original size, the effective error is reduced by nearly a factor of k.

Define an *expanded* image $\hat{\mathrm{I}}(x, y)$ of size $kN \times kN$ as follows:

$$\hat{\mathrm{I}}(x,y) = \begin{cases} I(\lfloor (x/k) + 1/2 \rfloor, \lfloor y/k \rfloor) & \text{if } y \equiv 0 \pmod{k}, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

The expanded image is stretched horizontally so that each data point is repeated in k consecutive columns. Vertically every kth row contains image data, while the rest are padded with zeros. Consider the application of the ADRT to this expanded image. Since each line sum uses exactly one data point from each row, the line sums $\hat{\mathrm{R}}(d, \theta)$, computed for $\hat{\mathrm{I}}(x, y)$, involve exactly one element from each row of the original image.

To complete the transform, we contract the Radon data by selecting every kth distance coordinate $d = j \cdot k, j = 0, 1, 2, \ldots, 2N - 1$. In addition, we must discard all but N different angles, selecting those which most closely match the ones produced by the algorithm on the original $N \times N$-sized images. Note that the angles computed on the expanded image do not, in general, contain those that would be computed in the initial image. Consider a line in I with x-displacement a. The slope of the corresponding line in $\hat{\mathrm{I}}$ should be approximately $-\frac{N-1}{a}$. Thus the line's x-displacement $\hat{\mathrm{a}}$ in $\hat{\mathrm{I}}$ should be such that $-\frac{kN-1}{\hat{\mathrm{a}}} = -\frac{N-1}{a}$, i.e., $\hat{\mathrm{a}} = \frac{kN-1}{N-1} \cdot a = (k + \frac{k-1}{N-1})a$. The closest such x-displacement available in $\hat{\mathrm{R}}$ is $(k + \lfloor \frac{k-1}{N-1} + \frac{1}{2} \rfloor)a$. Thus, if the lines are parameterized by their x-intercept d_x and x-displacement a, then contraction is defined as

$$\tilde{\mathrm{R}}(d_x, a) = \hat{\mathrm{R}}\left(kd, \left(k + \left\lfloor \frac{k-1}{N-1} + \frac{1}{2} \right\rfloor\right) a\right) \tag{4}$$

for $d_x = 0, 1, 2, \ldots, 2N - 1$, $a = 0, 1, 2, \ldots, N - 1$. The horizontal separation of this line from one of slope $-\frac{N-1}{a}$ is at most $1/2$ unit in $\hat{\mathrm{I}}$.

THEOREM 2. *Algorithm* 2 *can be used to produce a set of DRT data points* (d, θ) *for* $\theta = \tan^{-1}(a/(N-1))$, $a = 0, 1, 2, \ldots, N - 1$, *and* $d = d_x \cos\theta$, $d_x = 0, 1, 2, \ldots, 2N - 1$, *such that all pixels used to generate* (d, θ) *are within* $\frac{\lg kN}{k} + \frac{1}{2}$ *units of the line parameterized by* (d, θ). *The algorithm requires* $O(N^2k^2 \lg kN)$ *time on a sequential machine and* $O(\lg kN)$ *parallel steps using* $O(N^2k^2)$ *processors on an EREW PRAM.*

Proof. Consider a line L, parameterized by (d_x, a), to be sampled using the expand-ADRT-contract procedure described above with an expansion of factor k. We want to bound the horizontal distance between a point (x, y) on L from the corresponding sample point used in the ADRT. L corresponds to an "ideal" line $\hat{\mathrm{L}}$ of the same slope in the expanded image, parameterized by $(kd_x, (k + \frac{k-1}{N-1})a)$, and corresponding point (kx, ky) lies on $\hat{\mathrm{L}}$. From Theorem 1, the ADRT sample point for (kx, ky) in the expanded image is $\hat{\mathrm{I}}(kx + e, ky)$, where $|e| < \lg kN$. This is the value $I(\lfloor \frac{kx+e}{k} + \frac{1}{2} \rfloor, y)$ from the original image (see equation (3)). Therefore, the maximum possible error in the x-direction is bounded by $e/k + \frac{1}{2} \leq \frac{\lg kN}{k} + \frac{1}{2}$.

The time is dominated by the ADRT computation on the $kN \times kN$ expanded image. This is obtained by substituting kN for N in Theorem 1. ∎

Note that many of the computations in the expanded image are obviously unnecessary (e.g., segments consisting of padded zeroes, segments that will be discarded), and a careful implementation might greatly reduce the actual amount of computation. Furthermore, one might obtain a better approximation by interpolating the image data during the expansion rather than simply replicating it.

4. Approximate inverse of the DRT. The inverse of the RT is used in the reconstruction of medical images. A discrete version of Radon's inversion formula is as follows:

$$I(x,y) = \frac{\pi}{M} \sum_{i=1}^{M} Q(x\cos\theta_i + y\sin\theta_i, \theta_i). \tag{5}$$

$Q(d,\theta)$ represents the *filtered projection data*, which is computed from the Radon data $R(d,\theta)$ by convolving each row of data $R(d,\theta_i)$ with a filter $h(t)$, whose impulse response is $|\omega|$ in the frequency domain, where ω represents the frequency. A row of Q for fixed angle θ_i can be computed in the Fourier domain by

$$Q(d,\theta_i) = \text{FFT}^{-1}(\text{FFT}(R(d,\theta_i)) \cdot |\omega|). \tag{6}$$

This first step, which computes $Q(d,\theta)$, is called *filtering*, and the second step, represented by equation (5), is called *backprojection*. The computation of a single image pixel, $I(x,y)$, is done by summing the values of the M lines (one at each different angle) which pass through (x,y). This amounts to selecting one data point from each row, θ_i, whose d value is the closest to $x\cos\theta_i + y\sin\theta_i$. If both d and θ are uniformly sampled, then the M pixels used to compute $I(x,y)$ trace out a curve $d = x\cos\theta + y\sin\theta$ in discrete Radon space. To compute the complete image, we must sum along the family of sinusoidal curves $\{d = x\cos\theta + y\sin\theta\}$ for $x, y = 0, 1, 2, \ldots, N-1$.

Notice that the backprojection step is quite similar to the computation of the forward RT, except that points in image space trace out sinusoids in uniform Radon space rather than lines. Recall, however, that the data produced by the ADRT is not uniformly distributed in d or θ. (Angles in the range $0 - \pi/4$ are sampled at $\tan^{-1}(\frac{a}{N-1})$ for $a = 0, 1, 2, \ldots, N-1$, and distances are taken at $x\cos\theta_i$ for $x = 0, 1, 2, \ldots, N-1$.) It turns out that for Radon data sampled in this manner, points (x,y) trace out lines in the discrete Radon space. (More precisely, (x,y) traces out four separate lines corresponding to the ranges $\frac{\pi}{4}r \le \theta \le \frac{\pi}{4}(r+1)$, $r = 0, 1, 2, 3$.) Furthermore, the set of image points (x,y), when x and y are integers between 0 and $N-1$, trace out exactly the set of lines that are computed by the ADRT algorithm. Therefore, the ADRT algorithm can be used to compute an approximation to the backprojection of $Q(d,\theta)$. The algorithm is applied separately to each of the four groups of N angles, and the four resulting arrays are added to produce $I(x,y)$. (Note that angles $0, \pi/4$, $\pi/2$, and $3\pi/4$ are necessarily doubly represented in the set of four ranges of Radon data. One must therefore zero their values in one of the two ranges in which each appears before performing backprojection.) The following theorem summarizes the application of the ADRT to the group of angles $0 \le \theta \le \pi/4$.

THEOREM 3. *Given a set of filtered RT data points $Q(d,\theta)$ for $\theta = \tan^{-1}(a/(N-1))$, $a = 0, 1, 2, \ldots, N-1$, and $d = x\cos\theta$, $x = 0, 1, 2, \ldots, N-1$, Algorithm 2 can be used to produce the backprojection of the data $I(x,y), x, y = 0, 1, 2, \ldots, N-1$, such that all RT data points used to reconstruct pixel (x,y) are within $\frac{1}{2}(\lg N - 1)$ units of the line $d = x\cos\theta + y\sin\theta$. The algorithm requires $O(N^2 \lg N)$ time on a sequential machine and $O(\lg N)$ parallel steps using $O(N^2)$ processors on an EREW PRAM.*

Proof. The maximum error and execution times follow from the forward RT description in section 3; we must show that the lines computed by the ADRT correspond to the curves traced out by (x,y) in the nonuniformly sampled Radon space. First we show that the set of Radon space points (d,θ) corresponding to lines which pass through (x,y) is collinear. Recall that the angles are taken at $a = 0, 1, 2, \ldots, N-1$, where $\theta = \tan^{-1}(\frac{a}{N-1})$, so the angles are uniform in the x-displacement variable a.

Similarly, the distances are sampled at unit intervals horizontally. Letting h represent the horizontal distance from the origin of line (d, θ), we have $d = h \cos\theta$. Thus, in the (h, a) coordinate system, the samples represent $h, a = 0, 1, 2, \ldots, N-1$. Consider the pixel (x, y), which traces out the curve $d = x \cos\theta + y \sin\theta$ in the Radon data. Substituting for d, we obtain

$$h \cos\theta = x \cos\theta + y \sin\theta, \tag{7}$$

$$h = x + y\left(\frac{\sin\theta}{\cos\theta}\right) = x + y \tan\theta. \tag{8}$$

Since $\tan\theta = (\frac{a}{N-1})$, we have

$$h = x + \left(\frac{y}{N-1}\right) a, \tag{9}$$

which implies a line in (h, a) space (since x, y, and N are constant for a given image pixel).

Finally, we show that the lines defined in equation (9) are exactly the ones computed by the ADRT, by demonstrating that all such lines begin and end on grid points in (h, a). We need to show that in the top and bottom rows, $a = 0$ and $a = N-1$, the necessary h is an integer. Substituting into equation (9), at $a = 0$ we have $h = x$, and thus h is an integer in the range 0 to $N-1$. At $a = N-1$, $h = x + y$, so we have h an integer in the range 0 to $2N-1$. □

The quality of the approximation can be improved as before, by expanding and then contracting the Radon data set, as explained in section 3.4.

COROLLARY 1. *Given a set of filtered RT data points $Q(d, \theta)$ for $\theta = \tan^{-1}(a/(N-1))$, $a = 0, 1, 2, \ldots, N-1$, and $d = x \cos\theta$, $x = 0, 1, 2, \ldots, N-1$, Algorithm 2 can be used to produce the backprojection of the data $I(x, y), x, y = 0, 1, 2, \ldots, N-1$, such that all RT data points used to reconstruct pixel (x, y) are within $(\lg kN/k) + 1/2$ units of the line $d = x \cos\theta + y \sin\theta$. The algorithm requires $O(N^2k^2 \lg kN)$ time on a sequential machine and $O(\lg kN)$ parallel steps $O(N^2k^2)$ processors on an EREW PRAM.*

5. Conclusions. This paper presents a fundamentally new approach to computing the DRT and HT, taking advantage of the overlap in discrete lines to reduce the total number of computations. The ADRT algorithm yields an asymptotic factor of $N/\lg N$ speedup and due to its simplicity should lead to faster practical implementations as well. While the algorithm computes only an approximation to the desired transform, for realistic-sized images the maximum error is only a few units. This small error can be further reduced, with only a small factor increase in running time, by the expand/contract method described.

For appropriately sampled data (i.e., linograms), the ADRT can be directly applied to approximate the backprojection step of image reconstruction with $\Theta(N/lgN)$ speedup over standard methods. Since the filtering step can already be computed in $O(N^2 \lg N)$ time (in the Fourier domain, using the fast Fourier transform (FFT)), the entire image reconstruction computation time is reduced to $O(N^2 \lg N)$. An entirely different method which computes the image reconstruction from linograms in $O(N^2 \lg N)$ time has recently been discovered (see [6], [7]).

It is important to note that the general strategy of taking advantage of the overlap of near-parallel lines through discrete data can be applied to other important problems. For instance, three-dimensional volume visualization utilizes projections along

lines to compute two-dimensional views, and rotation of the views requires repeated computations along lines at neighboring angles. We are currently extending these ideas to develop fast approximation algorithms for volume visualization [18], [2].

Acknowledgments. The author has benefited from helpful discussion with Raghu Raghayan, Dave Swanson, and Tony Ticknor of Lockheed Missiles and Space Co., Inc., Palo Alto, California, during the course of this research.

REFERENCES

[1] D. H. Ballard, *Generalizing the Hough transform to detect arbitrary shapes*, Pattern Recognition, 10 (1978), pp. 129–143.

[2] M. Brady, W. Higgins, K. Ramaswamy, and R. Srinivasan, *Interactive navigation inside* $3D$ *radiological images*, in Proc. Biomedical Visualization '95, Atlanta, GA, IEEE Computer Society Press, Los Alamitos, CA, pp. 33–40; 85.

[3] M. Brady and W. Yong, *Fast parallel discrete approximation algorithms for the Radon transform*, in Proc. 4th ACM Symp. on Parallel Algorithms and Architectures, ACM, New York, 1992, pp. 91–99.

[4] R. E. Cypher, J. L. C. Sanz, and L. Snyder, *The Hough transform has* $O(N)$ *complexity on* $N \times N$ *mesh connected computers*, SIAM J. Comput., 19 (1990), pp. 805–820.

[5] R. O. Duda and P. E. Hart, *Use of the Hough transform to detect lines and curves in pictures*, Comm. ACM, 15 (1972), pp. 11–15.

[6] P. R. Edholm and G. T. Herman, *Linograms in image reconstruction from projections*, IEEE Trans. Medical Imaging, 6 (1987), pp. 301–307.

[7] P. R. Edholm and G. T. Herman, *Image reconstruction from linograms: Implementation and evaluation*, IEEE Trans. Medical Imaging, 7 (1988), pp. 239–246.

[8] C. Guerra and S. Hambrush, *Parallel algorithms for line detection on a mesh*, J. Parallel Distrib. Comput., 6 (1989), pp. 1–19.

[9] P. V. C. Hough, *Method and means of recognizing complex patterns*, U.S. Patent 3069654, 1962.

[10] J.-F. Jenq and S. Sahni, *Reconfigurable algorithms for the Hough transform*, in Proc. 1991 Intl. Conf. on Parallel Processing, St. Charles, IL, CRC Press, Boca Raton, FL, 1991, pp. 34–41.

[11] A. C. Kak, Guest ed., *Computerized Medical Images*, special issue of IEEE Trans. Biomed. Eng., BME-28 (Feb. 1981).

[12] J. Kittler and J. Illingworth, *A survey of the Hough transform*, Comput. Vision, Graphics Image Proc., 44 (1988), pp. 87–116.

[13] H. Liu, *Parallel Implementation of a Fast Radon Transform Algorithm*, M.S. thesis, Dept. of ECE, The Pennsylvania State University, University Park, PA, 1993.

[14] Y. Pan and Y. H. Chuang, *Parallel Hough transform algorithms on SIMD hypercube arrays*, in Proc. 1990 Intl. Conf. on Parallel Processing, St. Charles, IL, Penn. State Univ. Press, University Park, PA, 1990, pp. 83–86.

[15] J. Radon, *Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten*, Math.-Phys. Kl., 69 (1917), Berichte Saechsische Akademie der Wissenschaften, Leipzig, pp. 262–277.

[16] J. Ramakrishnan, *Approximate Fast Radon Transform Algorithm for Image Reconstruction—Implementation and Analysis*, M.S. Thesis, Dept. of ECE, The Pennsylvania State University, University Park, PA, 1993.

[17] J. L. C. Sanz and E. B. Hinkle, *Computing projections of digital images in image processing pipeline architectures*, IEEE Trans. ASSP, ASSP-35 (1987), pp. 198–207.

[18] T.-K. Wu and M. Brady, *Parallel approximate computation of projection for animated volume rendered displays*, in Proc. 1993 Parallel Rendering Symp., pp. 61–66; The Visual Computer, to appear.

SIAM J. COMPUT.
Vol. 27, No. 1, pp. 120–131, February 1998
© 1998 Society for Industrial and Applied Mathematics
006

VALUE SETS OF SOME POLYNOMIALS OVER FINITE FIELDS $GF(2^{2m})$*

THOMAS W. CUSICK†

Abstract. This paper shows that there is a connection between the crosscorrelation functions of certain binary m-sequences and the value sets of the polynomials $x^k(1+x)^{2^m-1}$ for $k \in \{\pm 1, \pm 2, 4\}$, where x is in the finite field $GF(2^{2m})$. In particular, the size of such value sets is determined by using finite field theory and known results about crosscorrelation functions.

Key words. finite field, polynomial, value set

AMS subject classifications. 68Q40, 11T06, 12E20

PII. S0097539794270352

1. Introduction.

Let $GF(q)$ denote the finite field with q elements, and let $f(x)$ be a polynomial of degree n in the ring $GF(q)[x]$. Define the value set $V(f)$ of $f(x)$ by

$$V(f) = \{f(x) : x \in GF(q)\}.$$

The problem of estimating the number of elements $|V(f)|$ in the value set was first given prominence by Chowla [2].

A polynomial $f(x)$ for which $|V(f)| = q$ is called a permutation polynomial, and such polynomials have been studied extensively (see Lidl and Niederreiter [6, pp. 347–368] for references up to 1983 and Mullen [7] for a survey of more recent work).

Birch and Swinnerton-Dyer [1] defined "general" polynomials over $GF(q)$ and proved that if $f(x)$ is such a polynomial of degree n, then

$$|V(f)| = q\left(\sum_{j=1}^{n} \frac{(-1)^{j-1}}{j!}\right) + 0\ (q^{1/2}).$$

Uchiyama [10] proved that if the degree n is at least 4 and $f(x)$ satisfies certain mild conditions, then $|V(f)| > \frac{1}{2}q$. Recent work on value sets develops these ideas (see Gomez-Calderon [3], Knopfmacher and Knopfmacher [5], Voloch [11], and von zur Gathen [12]), but the number of results on value sets is still relatively small.

The purpose of this paper is to show that for various values of k, the value sets of the special polynomials

$$f_m^{(k)}(x) = x^k(1+x)^{2^m-1}, \qquad x \in GF(2^{2m}),$$

can be counted for all positive integers m. If $k = 1$ or 2, use of the theory of the finite fields $GF(2^n)$ is enough to give the results. For other values of k, we also use known results on the crosscorrelation functions of certain binary m-sequences, which will be explained later. On the other hand, it turns out that results about certain value sets can be used to deduce new information about some crosscorrelation functions. This issue will be discussed in another paper.

*Received by the editors June 27, 1994; accepted for publication (in revised form) December 30, 1995. This research was supported by National Security Agency grant MDA904-94-H-2016.
http://www.siam.org/journals/sicomp/27-1/27035.html

†Department of Mathematics, State University of New York–Buffalo, Buffalo, NY 14214-3093 (cusick@acsu.buffalo.edu).

The polynomials considered in this paper have a special form, which is dictated by the methods of proof used below. However, there is reason to believe that the polynomials studied here are worthy of special attention, apart from the fact that their value sets can be counted. The case $k = 1$ (Theorem 1 below) provides the first examples of polynomials of arbitrarily large degree which attain an upper bound due to Wan [13] on the size of the value set of a polynomial which is not a permutation polynomial. Wan proved that if $f(x)$ is a polynomial of degree n over $GF(q)$ which is not a permutation polynomial, then

$$|V(f)| \le [q - n^{-1}(q-1)]$$

(here $[x]$ denotes the greatest integer not exceeding x). It follows from Theorem 1 that the polynomials $f_m^{(1)}(x)$ meet this bound when $q = 2^{2m}$.

We begin with the simplest cases, $k = 1$ and 2, where only finite field theory is used. It is convenient to define

$$S_m^{(k)} = \{x^k(1+x)^{2^m-1} : x \neq 0 \text{ or } 1, x \in GF(2^{2m})\}.$$

Of course this is just the value set $V(f_m^{(k)})$ with 0 omitted. We take $f_m^{(k)}(0) = 0$ even if k is negative, since $x^k = x^{2^n-1+k}$ in $GF(2^n)$.

2. Cases $k = 1$ and 2. For $k = 1$ and 2, we obtain not only a count of $S_m^{(k)}$ but also detailed information about the elements of these sets. We shall often use the fact that

$$(1+x)^{2^m-1} = \sum_{j=0}^{2^m-1} x^j \quad \text{for } x \in GF(2^{2m}), \tag{1}$$

which is true since all of the binomial coefficients $\binom{2^m-1}{j}$ are odd.

THEOREM 1. *Let τ generate the multiplicative group of nonzero elements of $GF(2^{2m})$. The sequence*

$$a_j = \tau^j(1+\tau^j)^{2^m-1}, \qquad 1 \le j \le 2^{2m} - 2,$$

is made up of 2^m elements 1, which occur when $2^m - 1$ divides j, and one occurrence of each of the $2^{2m} - 2^m - 2$ elements of $GF(2^{2m})$ which are nonzero and are not (2^m+1)st roots of one.

COROLLARY. *For each $m \ge 1$, $|S_m^{(1)}| = 2^{2m} - 2^m - 1$.*

Proof. First we show that for nonzero x and y in $GF(2^{2m})$, we can have

$$x(1+x)^{2^m-1} = y(1+y)^{2^m-1}, \qquad x \neq y, \tag{2}$$

only if x and y are (2^m+1)st roots of one. Taking the (2^m+1)st power in (2) gives

$$x^{2^m+1} = y^{2^m+1}. \tag{3}$$

Now (1), (2), and (3) imply

$$x + x^2 + \cdots + x^{2^m+1} = y + y^2 + \cdots + y^{2^m+1}$$

or (if $x \neq 1$ and $y \neq 1$)

$$x(x^{2^m+1}+1)(x+1)^{-1} = y(y^{2^m+1}+1)(y+1)^{-1}. \tag{4}$$

By (3), if $x^{2^m+1} \neq 1$ then (4) implies $x = y$.

Hence if (2) holds, then both x and y are (2^m+1)st roots of one. In that case we have (for $x \neq 1$)

$$x(1+x)^{2^m-1} = x(x^{2^m}+1)(x+1)^{-1} = (x^{2^m+1}+x)(x+1)^{-1} = 1.$$

Thus a 1 occurs in the sequence a_j exactly when τ^j is a (2^m+1)st root of one, i.e., when 2^m-1 divides j.

Finally, it is easy to see that if x is not a (2^m+1)st root of one, then neither is $x(1+x)^{2^m-1}$. This completes the proof of Theorem 1.

THEOREM 2. *Let τ generate the multiplicative group of nonzero elements of $GF(2^{2m})$. The sequence*

$$a_j = \tau^{2j}(1+\tau^j)^{2^m-1}, \quad 1 \le j \le 2^{2m}-2,$$

is made up of one occurrence of each of the 2^m (2^m+1)st roots of one other than 1; one occurrence of each of the 2^m-2 elements of $GF(2^m)$ other than 0 or 1; and two occurrences of those $2^{2m-1}-2^m$ elements of $GF(2^{2m})$ which are values of $f_m^{(2)}(x)$ for x not in $GF(2^m)$ and not a (2^m+1)st root of 1. In these last cases we have

$$f_m^{(2)}(x) = f_m^{(2)}(y) \quad \text{for } y = (x^{2^m+1}+x)(x+1)^{-1} \neq x.$$

COROLLARY. *For each $m \ge 1$, $|S_m^{(2)}| = 2^{2m-1}+2^m-2$.*

Proof. We must analyze the consequences of assuming

$$x^2(1+x)^{2^m-1} = y^2(1+y)^{2^m-1}, \qquad x \neq y. \tag{5}$$

Taking the (2^m+1)st power in (5) gives $x^{2(2^m+1)} = y^{2(2^m+1)}$ and so

$$x^{2^m+1} = y^{2^m+1}. \tag{6}$$

Now (1), (5), and (6) imply

$$x^2 + x^3 + \cdots + x^{2^m} = y^2 + y^3 + \cdots + y^{2^m}$$

or $(x^{2^m+1}+x^2)(x+1)^{-1} = (y^{2^m+1}+y^2)(y+1)^{-1}$. Crossmultiplying in this last equation and using (6) give

$$x^2 + y^2 = (x+y)x^{2^m+1} + (x+y)xy.$$

Now dividing by $x+y$ gives $x+y = x^{2^m+1}+xy$, so for $x \neq 1$ we have

$$y = (x^{2^m+1}+x)(x+1)^{-1} = x(x^{2^m}+1)(x+1)^{-1}. \tag{7}$$

Equation (7) is inconsistent with our assumption that $x \neq y$ if and only if $(x^{2^m}+1)(x+1)^{-1} = 1$, which is equivalent to $x^{2^m} = x$, i.e., $x \in GF(2^m)$. Since

$$f_m^{(2)}(x) = x^2(x^{2^m}+1)(x+1)^{-1} = x^2 \quad \text{for } x \in GF(2^m),$$

this shows that the 2^m-2 elements of $GF(2^m)$ other than 0 and 1 each occur once as values of a_j, when τ^j is in $GF(2^m)$.

We have assumed that (5) and (7) hold with $x \neq 1$, which implies $y \neq 1$. Thus the cases where $x^{2^m+1} = 1$ (and so $y = 1$ in (7)) are not covered. In these cases

$$f_m^{(2)}(x) = x(x^{2^m+1}+x)(x+1)^{-1} = x,$$

so the 2^m solutions $x \neq 1$ of $x^{2^m+1} = 1$ each occur once as values of a_j. This completes the proof of the assertions in Theorem 2.

To prove our results for other values of k, we use some known results on the cross-correlation functions for some binary m-sequences. The facts we need are developed in the next section.

3. Known results on crosscorrelation functions. A binary sequence generated by a linear recursion of order n and having the maximum period 2^n-1 is called a binary m-sequence (here "m" stands for "maximum period"). Crosscorrelation properties of such sequences have been studied extensively because of their importance in communications applications. In 1980, a survey of results in this area was given by Sarwate and Pursley [9].

It is well known that if $\{a_j : j = 1, 2, \ldots\}$ and $\{b_j : j = 1, 2, \ldots\}$ are two binary m-sequences with period 2^n-1, then there exists an integer d relatively prime to 2^n-1 and an integer t such that $b_{j+t} = a_{dj}$ for all j. Thus the study of the crosscorrelation between two binary m-sequences involves the functions

$$C_d(t) = \sum_{j=0}^{2^n-2} (-1)^{a_{j+t}+a_{dj}} \qquad (0 \le t \le 2^n - 2). \tag{8}$$

It will be convenient for us to define the crosscorrelation function $F_{d,n}(t) = F_d(t)$ of order n by

$$F_d(t) = C_d(t) + 1 \qquad (0 \le t \le 2^n - 2).$$

The set of values taken on by $F_d(t)$ for $0 \le t \le 2^n - 2$, together with a count of the number of times each values occurs, is called the crosscorrelation spectrum. It is well known (see [9, p. 602]) that the crosscorrelation spectrum depends only on d and not on the choice of the m-sequence $\{a_j\}$.

For later use, we introduce the trace function Tr_1^n from $GF(2^n)$ to $GF(2)$ by

$$Tr_1^n(x) = x + x^2 + x^{2^2} + \cdots + x^{2^{n-1}},$$

where $x \in GF(2^n)$. If the field $GF(2^n)$ is understood, we may write Tr in place of Tr_1^n. We remark that the crosscorrelation function is given by

$$F_{d,n}(t) = \sum_{x \in GF(2^n)} (-1)^{Tr(x\tau^t + x^d)}, \tag{9}$$

where τ generates the multiplicative group of nonzero elements of $GF(2^n)$ (to see this, take $a_j = Tr(\tau^j)$ in (8) and let $x = \tau^j$).

We prepare for the proof of our first theorem on crosscorrelation functions by giving three elementary lemmas.

LEMMA 1. *If α is an element of $GF(2^n)$, then*

$$\sum_{x \in GF(2^n)} (-1)^{Tr(\alpha x)} = \begin{cases} 0 & \text{if } \alpha \ne 0, \\ 2^n & \text{if } \alpha = 0. \end{cases}$$

Proof. The result follows from the fact that half of the elements of $GF(2^n)$ have trace 0 and the other half have trace 1.

LEMMA 2. *Every nonzero element x in $GF(2^{2m})$ can be represented uniquely as $x = \alpha\beta$, where α is in $GF(2^m)$ and β is a (2^m+1)st root of one in $GF(2^{2m})$.*

Proof. We let $\alpha = x^{2^{m-1}(2^m+1)}$ and $\beta = x^{2^{m-1}(2^m-1)}$.

LEMMA 3. *If α is in $GF(2^m)$ and β is in $GF(2^{2m})$, then*

$$Tr_1^{2m}(\alpha\beta) = Tr_1^m(\alpha(\beta + \beta^{2^m})).$$

Proof. We have

$$Tr_1^{2m}(\alpha\beta) = \sum_{j=0}^{2m-1} (\alpha\beta)^{2^j} = \sum_{j=0}^{m-1} (\alpha\beta)^{2^j} + \sum_{j=0}^{m-1} (\alpha\beta)^{2^{m+j}}$$
$$= \sum_{j=0}^{m-1} (\alpha(\beta+\beta^{2^m}))^{2^j} = Tr_1^m(\alpha(\beta+\beta^{2^m})).$$

The following theorem and proof are given in the thesis of Niho [8, pp. 40–41] under the supervision of L. R. Welch. Apparently the theorem was never published, so we repeat the proof here for the reader's convenience.

THEOREM 3. *Let τ generate the multiplicative group of nonzero elements of $GF(2^{2m})$. If $n = 2m$, d is relatively prime to $2^n - 1$, $d \equiv 2^r \bmod 2^m - 1$ for some $r (0 \le r \le m-1)$ and $d \equiv s \bmod 2^m + 1$, then*

$$F_{d,n}(t) = 2^m(N(\mu) - 1), \tag{10}$$

where $\mu = \tau^t$ and $N(\mu)$ is the number of distinct solutions x in $GF(2^{2m})$ to the system of equations

$$\mu x + x^{2^{-r}s} + \mu^{2^m} x^{-1} + x^{-2^{-r}s} = 0, \qquad x^{2^m+1} = 1. \tag{11}$$

Proof. Given $x \neq 0$ in $GF(2^{2m})$, let $x = \alpha\beta$ be the representation in Lemma 2; then we can write (9) as

$$F_{d,n}(t) = 1 + \sum_{\substack{\beta \in GF(2^{2m}) \\ \beta^{2^m+1}=1}} \sum_{\substack{\alpha \in GF(2^m) \\ \alpha \neq 0}} (-1)^{Tr(\alpha\beta\mu + \alpha^d \beta^d)}. \tag{12}$$

By our hypotheses, we have (using $Tr(x^2) = Tr(x)$)

$$Tr(\alpha\beta\mu + \alpha^d\beta^d) = Tr(\alpha\beta\mu + (\alpha^{2^r}\beta^s)^{2^{2m-r}}) = Tr(\alpha\beta\mu + \alpha\beta^{2^{-r}s}).$$

Substituting this into (12) and introducing terms with $\alpha = 0$ into the inner sum gives

$$F_{d,n}(t) = -2^m + \sum_{\substack{\beta \in GF(2^{2m}) \\ \beta^{2^m+1}=1}} \sum_{\alpha \in GF(2^m)} (-1)^{Tr(\alpha\beta\mu + \alpha\beta^{2^{-r}s})}. \tag{13}$$

By Lemma 3, the trace in (13) is

$$Tr_1^{2m}(\alpha(\beta\mu + \beta^{2^{-r}s})) = Tr_1^m(\alpha(\beta\mu + \beta^{2^{-r}s} + \mu^{2^m}\beta^{-1} + \beta^{-2^{-r}s})).$$

Substituting this into (13) and applying Lemma 1 to the inner sum give the desired formula (10).

LEMMA 4. *If d and d_1 are positive integers which satisfy $dd_1 \equiv 2^r \bmod 2^n - 1$ for some $r \ge 0$, then the functions $F_{d,n}(t)$ and $F_{d_1,n}(t)$ have the same crosscorrelation spectra.*

Proof. A computation gives $F_{d,n}(t) = F_{d_1,n}(-dt)$.

Later we shall require the following theorems which evaluate particular crosscorrelation spectra. For these theorems, it is convenient to extend the domain of $F_{d,n}(t)$ to all of $GF(2^n)$ by defining

$$F_{d,n}(-\infty) = 0.$$

TABLE 1

$N(\mu)$	Value of $F_d(t)$	Number of times given value occurs: m odd	m even
4	$3 \cdot 2^m$	$\frac{1}{3}\,(2^{2m-3} - 2^{m-2})$	$\frac{1}{3}\,(2^{2m-3} - 2^{m-1})$
3	2^{m+1}	2^{m-1}	2^{m-1}
2	2^m	$2^{2m-2} - 2^{m-1}$	2^{2m-2}
1	0	$\frac{1}{3}\,(2^{2m} + 5 \cdot 2^{m-1})$	$\frac{1}{3}\,(2^{2m} + 2^{m-1})$
0	-2^m	$3(2^{2m-3} - 2^{m-2})$	$3 \cdot 2^{2m-3} - 2^{m-1}$

TABLE 2

$N(\mu)$	Value of $F_d(t)$	Number of times given value occurs
3	2^{m+1}	$\frac{1}{3}\,(2^{2m-1} - 2^{m-1})$
2	2^m	2^m
1	0	$2^{2m-1} - 2^{m-1}$
0	-2^m	$\frac{1}{3}\,(2^{2m} - 2^m)$

THEOREM 4. *Let $n = 2m$ with $m \geq 2$, and let $d = 2^m + 3$. Then the crosscorrelation spectrum for $F_{d,n}(t)$ is given by Table* 1. *Here $N(\mu)$ is the number of solutions of the system* (11) *with $r = 2 = s$.*

Proof. Niho [8, Thm. 3.8, p. 51] proved that when $d = 2^m+3$, $N(\mu) \leq 4$ must hold in Theorem 3 above. Niho [8, p. 55] also conjectured the crosscorrelation spectrum as given above, and Helleseth [4, Thm. 4.8, pp. 221–222] proved this conjecture. Note that for $m = 2$, $F_d(t) = 12$ occurs zero times. In every other case $F_d(t)$ actually takes on five different values.

THEOREM 5. *Let $n = 2m$ with m even and $m \geq 2$. Let $d = 2^{m+1} - 1$. Then the crosscorrelation spectrum for $F_{d,n}(t)$ is given by Table* 2. *Here $N(\mu)$ is the number of solutions of the system* (11) *with $r = 0$ and $s = -3$.*

Proof. This is a result of Niho [8, Thm. 3.6, p. 43].

We remark that the proofs of Theorems 4 and 5 involve detailed analyses of the solutions of certain equations over finite fields.

4. The cases $k = -2$ and 4. For $k = -2$ and 4, we are unable to obtain the kind of specific information about the elements of the sets $S_m^{(k)}$ that was achievable for $k = 1$ and 2 (see section 2 above); however, we can obtain an exact count for these sets $S_m^{(k)}$.

THEOREM 6. *For $k = -2$ and 4, we have for each $m \geq 2$*

$$|S_m^{(-2)}| = |S_m^{(4)}| = 5 \cdot 2^{2m-3} + 2^{m-1} - 2 \quad \textit{if } m \textit{ is even}$$

and

$$|S_m^{(-2)}| = |S_m^{(4)}| = 5 \cdot 2^{2m-3} + \frac{1}{3}(2^{m-2} - 2) - 1 \quad \textit{if } m \textit{ is odd.}$$

For later reference, it is convenient to have the following trivial lemma.

LEMMA 5. *If $\beta \in GF(2^{2m})$ and $\beta^{2^m+1} = 1$, then $\beta(1+\beta)^{2^m-1} = 1$.*

Proof. We have

$$\beta(1+\beta)^{2^m-1} = \beta(\beta^{2^m}+1)(\beta+1)^{-1} = 1,$$

where the first equality follows from (1).

In proving Theorem 6, we shall deal first with the case $k = 4$. We take $d = 3\cdot 2^m - 1$ in Theorem 3, so $r = 1$, $s = -4$, and the system (11) becomes

$$\mu x + x^{-2} + \mu^{2^m} x^{-1} + x^2 = 0, \qquad x^{2^m+1} = 1$$

or

$$x^4 + \mu x^3 + \mu^{2^m} x + 1 = 0, \qquad x^{2^m+1} = 1. \tag{14}$$

Suppose that (14) has a solution $x = \beta$ for a given μ, and define γ by $\mu = \beta\gamma$. Then, provided that $\gamma \neq 1$, the first equation in (14) gives, using (1),

$$\beta^4 = (\gamma^{2^m}+1)(\gamma+1)^{-1} = (1+\gamma)^{2^m-1},$$

so

$$\mu^4 = \beta^4\gamma^4 = \gamma^4(1+\gamma)^{2^m-1} \in S_m^{(4)}.$$

Conversely, given $\gamma \in GF(2^{2m})$ we can define μ by

$$\mu^4 = \gamma^4(1+\gamma)^{2^m-1} \in S_m^{(4)}$$

and β by $\beta = \mu/\gamma$. Then, provided $\gamma \neq 1$,

$$\beta^4 = (1+\gamma)^{2^m-1} = (\gamma^{2^m}+1)(\gamma+1)^{-1},$$

and the system (14) is satisfied by $x = \beta$, because $(1+\gamma)^{2^m-1}$ (and hence its unique fourth root β) is clearly a (2^m+1)st root of 1 in $GF(2^{2m})$. If we define

$$H_m = \{\mu : \mu \in GF(2^{2m}) \text{ and the system (14) has at least one solution}\},$$

then we can establish the following lemma.

LEMMA 6. *For each $m \geq 2$,*

$$|H_m| - |S_m^{(4)}| = 2 \quad \textit{if } m \textit{ is even}$$

and

$$|H_m| - |S_m^{(4)}| = \frac{1}{3}(2^{m+1}+2) + 1 \quad \textit{if } m \textit{ is odd}.$$

Proof. By the argument in the paragraph preceding the lemma, there is a one-to-one correspondence between H_m and $S_m^{(4)}$ except that 0, and possibly some elements $\mu = \beta\gamma$ in H_m which have $\gamma = 1$, are missing from $S_m^{(4)}$. In fact we have shown that, apart from the missing elements just mentioned, $S_m^{(4)}$ is made up of fourth powers of elements of H_m. Thus, the elements of H_m that are not in $S_m^{(4)}$ are 0 and also those (2^m+1)st roots of 1 which are not in $S_m^{(4)}$.

If β is a (2^m+1)st root of 1 and $f_m^{(4)}(x) = \beta$, then x is also a (2^m+1)st root of 1. Also, by Lemma 5

$$\beta^{2^m+1} = 1 \text{ implies } \beta^4(1+\beta)^{2^m-1} = \beta^3.$$

Hence

$$S_m^{(4)} \cap \{\beta : \beta^{2^m+1} = 1\} = \{\beta^3 : \beta^{2^m+1} = 1\}. \tag{15}$$

Since the set on the right-hand side in (15) is $\{\beta : \beta^{(2^m+1)/3} = 1\}$, this proves Lemma 6 for m odd (the elements of H_m not in $S_m^{(4)}$ are 0 and the $(2^{m+1}+2)/3$ (2^m+1)st roots of 1 which are not in the set just mentioned). For m even, it is clear that 1 is not in $S_m^{(4)}$, so the set on the left-hand side of (15) is simply the set of all (2^m+1)st roots of 1 except 1. This proves Lemma 6 for m even (the elements of H_m not in $S_m^{(4)}$ are just 0 and 1).

The system (14) was obtained by taking $d = 3 \cdot 2^m - 1$ in Theorem 3. Since

$$(3 \cdot 2^m - 1)(2^m + 3) \equiv 2^{m+3} \bmod 2^{2m} - 1,$$

it follows from Lemma 4 that Table 1 is also valid for $d = 3 \cdot 2^m - 1$. Hence $|H_m|$ in Lemma 6 is simply 2^{2m} minus the entry for $N(\mu) = 0$ in Table 1. If we plug in those values for $|H_m|$, Lemma 6 immediately gives Theorem 6 for $k = 4$.

Now we turn to the case $k = -2$ of Theorem 6. We take $d = 2^m + 3$ in Theorem 3, so $r = 2$, $s = 2$, and the system (11) becomes

$$\mu x + x^{\frac{1}{2}} + \mu^{2m} x^{-1} + x^{-\frac{1}{2}} = 0, \qquad x^{2^m+1} = 1$$

or (squaring the first equation and changing notation by replacing μ^2 by μ)

$$\mu x^4 + x^3 + x + \mu^{2^m} = 0, \qquad x^{2^m+1} = 1. \tag{16}$$

Suppose that (16) has a solution $x = \beta$ for a given μ, and define γ by $\mu = \beta^{-1}\gamma$. Then, provided that $\gamma \neq 1$, the first equation in (16) gives, using (1),

$$\beta^2 = (\gamma^{2^m} + 1)(\gamma + 1)^{-1} = (1 + \gamma)^{2^m - 1},$$

so

$$\mu^{-2} = \beta^2 \gamma^{-2} = \gamma^{-2}(1 + \gamma)^{2^m - 1} \in S_m^{(-2)}.$$

If we define

$$G_m = \{\mu : \mu \in GF(2^{2m}) \text{ and the system (16) has at least one solution}\},$$

then by the above remarks there is a one-to-one correspondence between G_m and $S_m^{(-2)}$ except that 0, and possibly some elements $\mu = \beta^{-1}\gamma$ in G_m which have $\gamma = 1$, are missing from $S_m^{(-2)}$. In fact apart from the missing elements, $S_m^{(-2)}$ is made up of squares of reciprocals of elements of G_m. This enables us to prove the following lemma.

LEMMA 7. *For each $m \geq 2$,*

$$|G_m| - |S_m^{(-2)}| = 2 \quad \textit{if } m \textit{ is even}$$

and

$$|G_m| - |S_m^{(-2)}| = \frac{1}{3}\,(2^{m+1} + 2) + 1 \quad \textit{if } m \textit{ is odd}.$$

Proof. The argument is very similar to the proof of Lemma 6. By Lemma 5,

$$\beta^{2^m+1} = 1 \text{ implies } \beta^{-2}(1+\beta)^{2^m-1} = \beta^{-3}.$$

Hence (15) holds with $S_m^{(-2)}$ in place of $S_m^{(4)}$. Now the proof of Lemma 7 is completed as in the proof of Lemma 6.

The system (16) was obtained by taking $d = 2^m + 3$ in Theorem 3. Hence, as before we can evaluate G_m by using Theorem 4 and we see that $|G_m| = |H_m|$. Therefore, by Lemmas 6 and 7 we have $|S_m^{(-2)}| = |S_m^{(4)}|$ for $m \geq 2$ and the proof of Theorem 6 is complete.

5. The cases $k = -1$ and 3. We are able to get an exact count of $S_m^{(-1)}$ when m is even. The corresponding problem with m odd seems to be more difficult, for reasons which we will examine below.

THEOREM 7. *For each even $m \geq 2$, we have*

$$|S_m^{(-1)}| = \frac{1}{3}\,(2^{2m+1} + 2^m) - 2.$$

To begin the proof of Theorem 7, we take $d = 2^{m+1} - 1$ in Theorem 3 (note d is relatively prime to $2^{2m} - 1$ since m is even), so $r = 0$, $s = -3$, and the system (11) becomes

$$\mu x + x^{-3} + \mu^{2^m} x^{-1} + x^3 = 0, \qquad x^{2^m+1} = 1$$

or (multiplying the first equation by x^3, changing notation by replacing μ by μ^2, and taking the square root of the first equation)

$$x^3 + \mu x^2 + \mu^{2^m} x + 1 = 0, \qquad x^{2^m+1} = 1. \tag{17}$$

We define

$$g_\mu(x) = g(x) = x^3 + \mu x^2 + \mu^{2^m} x + 1 \qquad (\mu \in GF(2^{2m}))$$

and

$$h_\xi(x) = h(x) = \xi x^3 + x^2 + x + \xi^{2^m} \qquad (\xi \in GF(2^{2m})).$$

Our next lemma shows that when m is even, the system (17) involving $g_\mu(x)$ and the system

$$\xi x^3 + x^2 + x + \xi^{2^m} = 0, \quad x^{2^m+1} = 1 \tag{18}$$

involving $h_\xi(x)$ have identical patterns of numbers of solutions. To state the lemma, we define

$$N_j^{(g)}(m) = N_j^{(g)} = |\{\mu \in GF(2^{2m}) : (17) \text{ has exactly } j \text{ solutions } x \in GF(2^{2m})\}|$$

and

$$N_j^{(h)}(m) = N_j^{(h)} = |\{\xi \in GF(2^{2m}) : (18) \text{ has exactly } j \text{ solutions } x \in GF(2^{2m})\}|.$$

LEMMA 8. *If m is even, then $N_j^{(g)}(m) = N_j^{(h)}(m)$ for all j, and $N_j^{(g)}(m)$ is nonzero if and only if $0 \leq j \leq 3$.*

Proof. If we let $x = \mu^{2^m-1}y$ ($\mu \neq 0$) in $g_\mu(x)$, then we obtain

$$\mu^{2^m-2}y^3 + y^2 + y + \mu^{-2^{m+1}+1} = 0.$$

Since $(\mu^{2^m-2})^{2^m} = \mu^{-2^{m+1}+1}$, we can define $\xi = \mu^{2^m-2}$ and get

$$h_\xi(y) = \xi y^3 + y^2 + y + \xi^{2^m} = 0.$$

Since $x^{2^m+1} = (\mu^{2^m-1}y)^{2^m+1} = y^{2^m+1}$ and

$$gcd(2^m - 2, 2^{2m} - 1) = \begin{cases} 1, m \text{ even}, \\ 3, m \text{ odd}, \end{cases}$$

for m even $\xi = \mu^{2^m-2}$ gives a one-to-one correspondence between ξ and μ, and the systems (17) and (18) have exactly the same number of solutions except possibly when $\mu = 0$ or $\xi = 0$. It is easy to see that for m even, (17) and (18) each have exactly one solution ($x = 1$) when $\mu = \xi = 0$. Thus $N_j^{(g)}(m) = N_j^{(h)}(m)$ holds for all j when m is even.

For the final assertion in the lemma we appeal to Theorem 5, which says that $N_j^{(g)}(m)$ is nonzero for m even if and only if $0 \leq j \leq 3$.

Now we can complete the proof of Theorem 7. Suppose that (18) has a solution $x = \beta$ for a given ξ, and define γ by $\xi = \beta^{-1}\gamma$. Then, provided that $\gamma \neq 1$, the first equation in (18) gives, using (1),

$$\beta = (\gamma^{2^m} + 1)(\gamma + 1)^{-1} = (1 + \gamma)^{2^m-1},$$

so

$$\xi^{-1} = \beta\gamma^{-1} = \gamma^{-1}(1 + \gamma)^{2^m-1} \in S_m^{(-1)}.$$

If we define

$$I_m = \{\xi : \xi \in GF(2^{2m}) \text{ and the system (18) has at least one solution}\},$$

then by the above remarks there is a one-to-one correspondence between I_m and $S_m^{(-1)}$ except that 0, 1, and possibly some other elements $\xi = \beta^{-1}\gamma$ in I_m which have $\gamma = 1$, are missing from $S_m^{(-1)}$. In fact, we can prove in our next lemma that the only missing elements are 0 and 1.

LEMMA 9. *For each $m \geq 1$,*

$$|I_m| - |S_m^{(-1)}| = 2.$$

Proof. The argument is very similar to the proofs of Lemmas 6 and 7. Here there is no distinction between m even and m odd, because Lemma 5 gives

$$\beta^{2^m+1} = 1 \text{ implies } \beta^{-1}(1 + \beta)^{2^m-1} = \beta^{-2};$$

hence for all m, $S_m^{(-1)}$ contains all of the $(2^m + 1)$st roots of 1 except 1.

We know from Theorem 5 that for m even

$$|I_m| = 2^{2m} - N_0^{(g)}(m) = 2^{2m} - \frac{1}{3}\,(2^{2m} - 2^m).$$

TABLE 3

i	$N_i^{(g)}(m)$	$N_i^{(h)}(m)$
3	$\frac{1}{3}\,(2^{2m-1}-2^{m-1}+2)$	$\frac{1}{3}\,(2^{2m-1}-2^{m-1}-1)$
2	2^m-2	2^m
1	$2^{2m-1}-2^{m-1}+2$	$2^{2m-1}-2^{m-1}+1$
0	$\frac{1}{3}\,(2^{2m}-2^m-2)$	$\frac{1}{3}\,(2^{2m}-2^m-2)$

Now Lemma 9 gives Theorem 7.

We make the following conjecture.

CONJECTURE 1. *For each odd $m \geq 1$, we have Table 3.*

By Lemma 9, Conjecture 1 implies the following conjecture.

CONJECTURE 2. *For each odd $m \geq 1$, we have*

$$|S_m^{(-1)}| = \frac{1}{3}\,(2^{2m+1}+2^m-4).$$

Of course, to prove Conjecture 2 it would be enough to prove the formula for $N_0^{(g)}(m)$ given in Conjecture 1; we would not need the rest of Conjecture 1. Unfortunately, we can prove only the following lemma.

LEMMA 10. *For each odd $m \geq 1$, we have*

$$N_2^{(g)}(m) = 2^m - 2 \quad \textit{and} \quad N_2^{(h)}(m) = 2^m.$$

Proof. Niho [8, pp. 44–46] proved $N_2^{(g)}(m) = 2^m$ for even $m \geq 2$ by a direct count of the number of μ for which $g_\mu(x) = 0$ can have a repeated root of multiplicity 2. Analogous arguments suffice to prove Lemma 10. We omit the rather tedious details.

We can also prove that if m is odd, then 3 divides $N_3^{(g)}(m) - 1$ and $N_i^{(g)}(m)$ for $i = 0, 1, 2$ (all this follows from Conjecture 1); we omit the details since we make no use of the result.

Now we turn to the case $k = 3$, where we have the following conjecture.

CONJECTURE 3. *For each odd $m \geq 1$, $|S_m^{(3)}| = |S_m^{(-1)}|$.*

We show how this conjecture can be reduced to

$$N_0^{(g)}(m) = N_0^{(h)}(m) \quad \text{for odd } m \geq 1, \tag{19}$$

which is part of Conjecture 1.

Suppose that (17) has a solution $x = \beta$ for a given μ and define γ by $\mu = \beta\gamma$. Then, provided that $\gamma \neq 1$, the first equation in (17) gives, using (1),

$$\beta^3 = (\gamma^{2^m}+1)(\gamma+1)^{-1} = (1+\gamma)^{2^m-1},$$

so

$$\mu^3 = \beta^3\gamma^3 = \gamma^3(1+\gamma)^{2^m-1} \in S_m^{(3)}.$$

If we define

$$J_m = \{\mu : \mu \in GF(2^{2m}) \text{ and the system (17) has at least one solution } x\},$$

then we can prove

$$|J_m| - |S_m^{(3)}| = 2 \quad \text{for odd } m \geq 1. \tag{20}$$

This follows because for m odd, $\mu^3 = \beta^3\gamma^3$ has a unique cube root $\mu = \beta\gamma$ in $GF(2^{2m})$. Therefore (as in the proofs of Lemmas 6 and 7) there is a one-to-one correspondence between J_m and $S_m^{(3)}$ (made up of cubes of elements of J_m), except that 0 and 1 are in J_m but not in $S_m^{(3)}$.

If (19) is true, then $|I_m| = |J_m|$ for odd $m \geq 1$. Now Conjecture 3 follows from (20) and Lemma 9.

Conjecture 3 is certainly false for even m, since then 3 divides $2^m - 1$ and the one-to-one correspondence used to prove (20) is certainly false. For m even, every element $\gamma^3(1+\gamma)^{2^m-1}$ of $S_m^{(3)}$ is a $\frac{1}{3}(2^{2m}-1)$st root of 1 in $GF(2^{2m})$. It is reasonable to conjecture that all of these roots of 1 are actually in $S_m^{(3)}$, which gives our last conjecture.

CONJECTURE 4. *For each even $m \geq 2$, $|S_m^{(3)}| = \frac{1}{3}(2^{2m} - 1)$.*

REFERENCES

[1] B. J. BIRCH AND H. P. F. SWINNERTON-DYER, *Note on a problem of Chowla*, Acta Arith., 5 (1959), pp. 417–423.

[2] S. CHOWLA, *The Riemann zeta function and allied functions*, Bull. Amer. Math. Soc., 58 (1952), pp. 287–305.

[3] J. GOMEZ-CALDERON, *On the cardinality of value set of polynomials with coefficients in a finite field*, Proc. Japan Acad. Ser. A Math. Sci., 68 (1992), pp. 338–340.

[4] T. HELLESETH, *Some results about the cross-correlation function between two maximal linear sequences*, Discrete Math., 16 (1976), pp. 209–232.

[5] A. KNOPFMACHER AND J. KNOPFMACHER, *The distribution of values of polynomials over a finite field*, Linear Algebra Appl., 134 (1990), pp. 145–151.

[6] R. LIDL AND H. NIEDERREITER, *Finite Fields*, Addison–Wesley, Reading, MA, 1983.

[7] G. L. MULLEN, *Permutation polynomials over finite fields*, in Finite Fields, Coding Theory and Advances in Communications and Computing, G. L. Mullen and P. Shiue, eds., Marcel Dekker, New York, 1993, pp. 131–151.

[8] Y. NIHO, *Multi-valued Cross-correlation Functions Between Two Maximal Linear Recursive Sequences*, Ph.D. thesis, USCEE Report 409, Dept. Elec. Eng., Univ. Southern California, Los Angeles, CA, 1972.

[9] D. V. SARWATE AND M. B. PURSLEY, *Crosscorrelation properties of pseudorandom and related sequences*, Proc. IEEE, 68 (1980), pp. 593–619.

[10] S. UCHIYAMA, *Sur le nombre des valeurs distinctes d'un polynome a coefficients dans un corps fini*, Proc. Japan. Acad., 30 (1954), pp. 930–933.

[11] J. P. VOLOCH, *On the number of values taken by a polynomial over a finite field*, Acta Arith., 52 (1989), pp. 197–201.

[12] J. VON ZUR GATHEN, *Values of polynomials over finite fields*, Bull. Austral. Math. Soc., 43 (1991), pp. 141–146.

[13] D. WAN, *A p-adic lifting lemma and its applications to permutation polynomials*, in Finite Fields, Coding Theory and Advances in Communications and Computing, G. L. Mullen and P. Shiue, eds., Marcel Dekker, New York, 1993, pp. 209–216.

SIAM J. COMPUT.
Vol. 27, No. 1, pp. 132–169, February 1998
© 1998 Society for Industrial and Applied Mathematics
007

OPTIMAL UPWARD PLANARITY TESTING OF SINGLE-SOURCE DIGRAPHS*

PAOLA BERTOLAZZI†, GIUSEPPE DI BATTISTA‡, CARLO MANNINO§, AND ROBERTO TAMASSIA¶

Abstract. A digraph is upward planar if it has a planar drawing such that all the edges are monotone with respect to the vertical direction. Testing upward planarity and constructing upward planar drawings is important for displaying hierarchical network structures, which frequently arise in software engineering, project management, and visual languages. In this paper we investigate upward planarity testing of single-source digraphs; we provide a new combinatorial characterization of upward planarity and give an optimal algorithm for upward planarity testing. Our algorithm tests whether a single-source digraph with n vertices is upward planar in $O(n)$ sequential time, and in $O(\log n)$ time on a CRCW PRAM with $n \log\log n / \log n$ processors, using $O(n)$ space. The algorithm also constructs an upward planar drawing if the test is successful. The previously known best result is an $O(n^2)$-time algorithm by Hutton and Lubiw [*Proc. 2nd ACM–SIAM Symposium on Discrete Algorithms*, SIAM, Philadelphia, 1991, pp. 203–211]. No efficient parallel algorithms for upward planarity testing were previously known.

Key words. graph drawing, planar graph, upward drawing, triconnected components, parallel algorithm

AMS subject classifications. 68Q20, 68R10, 68U05, 05C10, 06A99

PII. S0097539794279626

1. Introduction.

The upward planarity of digraphs is a fundamental issue in the area of graph drawing and has been extensively investigated. A digraph is upward planar if it has a planar upward drawing, i.e., a planar drawing such that all the edges are monotone with respect to the vertical direction (see Figure 1a). Planarity and acyclicity are necessary but not sufficient conditions for upward planarity, as shown in Figure 1b.

Testing upward planarity and constructing upward planar drawings are important for displaying hierarchical network structures, which frequently arise in a wide variety of areas. Key areas of application include software engineering, project management, and visual languages. Especially significant in a number of applications are single-source digraphs, such as subroutine-call graphs, is-a hierarchies, and organization charts. Also, upward planarity of single-source digraphs has deep combinatorial

*Received by the editors December 6, 1994; accepted for publication (in revised form) January 4, 1996. An extended abstract of this paper was presented at the *First European Symposium on Algorithms* (ESA '93), Bonn, Germany, Lecture Notes in Comput. Sci. 726, Springer-Verlag, New York, 1993, pp. 37–48. This research was performed in part while Roberto Tamassia was visiting IASI–CNR. This research was supported in part by the National Science Foundation under grant CCR-9423847, by the NATO Scientific Affairs Division under collaborative research grant 911016, by the Italian National Research Council (CNR) under Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo, Sottoprogetto 6, Infokit and under grant 95.00459.CT12, and by the ESPRIT II Basic Research Actions Program of the European Community (project Algorithms and Complexity).
http://www.siam.org/journals/sicomp/27-1/27962.html

†IASI–CNR, Viale Manzoni, 30, 00185 Roma, Italy (bertola@iasi.rm.cnr.it).

‡Dipartimento di Informatica e Automazione, Università degli Studi di Roma Tre, Via della Vasca Navale, 79, 00146 Roma, Italy (dibattista@iasi.rm.cnr.it).

§IASI–CNR, Viale Manzoni, 30, 00185 Roma, Italy and Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza," Via Buonarroti, 12, 00185 Roma, Italy (mannino@iasi.rm.cnr.it).

¶Department of Computer Science, Brown University, Providence, RI 02912-1910 (rt@cs.brown.edu).

implications in the theory of ordered sets. Namely, the orders defined by the transitive closure of upward planar single-source digraphs have bounded dimension [34] so that they can be compactly represented.

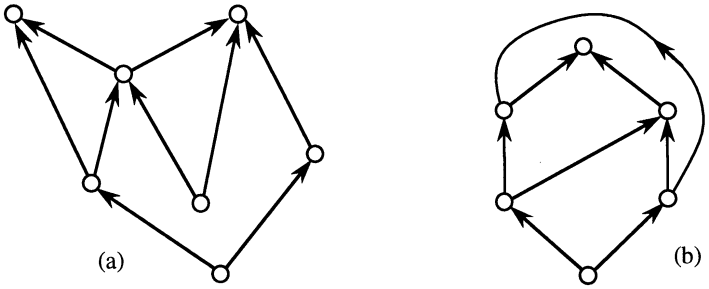


FIG. 1. *Examples of planar acyclic digraphs:* (a) *upward planar;* (b) *not upward planar.*

A survey on algorithms for planarity testing and graph drawing can be found in [7]. Previous work on upward planarity is as follows.

Combinatorial results on upward planarity for covering digraphs of lattices were first given in [22, 26]. Further results on the interplay between upward planarity and ordered sets are surveyed by Rival [30]. Lempel, Even, and Cederbaum [23] relate the planarity of biconnected undirected graphs to the upward planarity of st-digraphs. A combinatorial characterization of upward planar digraphs is provided in [21, 9]; namely, a digraph is upward planar if and only if it is a subgraph of a planar st-digraph.

Di Battista, Tamassia, and Tollis [9, 12] give algorithms for constructing upward planar drawings of st-digraphs and investigate area bounds and symmetry display. Tamassia and Vitter [32] show that the above drawing algorithms can be efficiently parallelized. Upward planar drawings of trees and series-parallel digraphs are studied in [29, 31, 6, 13, 15] and [1, 2], respectively.

In [8] it is shown that for the special case of bipartite digraphs, upward planarity is equivalent to planarity. In [3, 4] a polynomial-time algorithm is given for testing the upward planarity of digraphs with a prescribed embedding. Thomassen [33] characterizes the upward planarity of single-source digraphs in terms of forbidden circuits. Hutton and Lubiw [19] combine Thomassen's characterization with a decomposition scheme to test the upward planarity of an n-vertex single-source digraph in $O(n^2)$ time. Very recently, Papakostas [25] has given a polynomial-time algorithm for upward planarity testing of outerplanar digraphs, and Garg and Tamassia [16] have shown that upward planarity testing is NP-complete for general digraphs.

In this paper we investigate upward planarity testing of single-source digraphs. Our main results are summarized as follows:

- We provide a new combinatorial characterization of upward planarity within a given embedding in terms of a forest embedded in the face-vertex incidence graph.
- We reduce the upward planarity testing problem to that of finding a suitable orientation of a tree that synthetically represents the decomposition of a graph into its triconnected components.
- We show that the above combinatorial results yield an optimal $O(n)$-time upward planarity testing algorithm for single-source digraphs. The algorithm also constructs an upward planar drawing if the test is successful. Our algorithm is an improvement over the previously known best result [19] by an $O(n)$ factor in the time

complexity. Our algorithm is easy to implement and does not require any complex data structure.

- We efficiently parallelize the above algorithm to achieve $O(\log n)$ time on a CRCW PRAM with $n \log\log n / \log n$ processors. Hence, we provide the first efficient parallel algorithm for upward planarity testing. Our parallel time complexity is the same as that of the best parallel algorithm for planarity testing [28, 27].
- Finally, as a side effect we provide an optimal parallel algorithm for testing acyclicity of a planar n-vertex single-source digraph in $O(\log n)$ time with $n/\log n$ processors on an EREW PRAM.

Open problems include the following:

- devising efficient dynamic algorithms for upward planarity testing of single-source digraphs;
- exploring the area requirements of upward planar drawings of single-source digraphs;
- reducing the time complexity of upward planarity testing of planar digraphs with a prescribed embedding; and
- identifying additional classes of planar digraphs for which upward planarity can be tested in polynomial time.

The remainder of this paper is organized as follows. Section 2 contains preliminary definitions and results. The problem of testing upward planarity for planar single-source digraphs with a prescribed embedding is investigated in section 3. A combinatorial characterization of upward planarity for single-source digraphs is given in sections 4, 5, and 6. The complete upward planarity testing algorithm for single-source digraphs is presented in section 7. Also in section 7, two examples of application of the algorithm are illustrated. In the first example the considered digraph is not upward drawable; in the second example an upward drawable digraph is considered.

2. Preliminaries. In this section we recall some terminology and basic results on upward planarity. We also review the SPQR-tree, introduced in [10, 11], and the combinatorial characterization of upward planarity for embedded planar digraphs, shown in [3, 4]. We assume the reader's familiarity with planar graphs.

2.1. Drawings and embeddings. A drawing of a graph maps each vertex to a distinct point of the plane and each edge (u, v) to a simple Jordan curve with endpoints u and v. A *polyline* drawing maps each edge into a polygonal chain. A *straight-line* drawing maps each edge into a straight-line segment.

A drawing is planar if no two edges intersect except, possibly, at common endpoints. A graph is planar if it has a planar drawing. Two planar drawings of a planar graph G are equivalent if, for each vertex v, they have the same circular clockwise sequence of edges incident on v. Hence, the planar drawings of G are partitioned into equivalence classes. Each such class is called an *embedding* of G. An embedded planar graph is a planar graph with a prescribed embedding. A triconnected planar graph has a unique embedding up to a reflection. A planar drawing divides the plane into topologically connected regions delimited by circuits, called faces. The external face is the boundary of the unbounded region. Two drawings with the same embedding have the same faces. Hence, one can speak of the faces of an embedding.

Let G be a digraph, i.e., a directed graph. A *source* of G is a vertex without incoming edges. A *sink* of G is a vertex without outgoing edges. An *internal vertex* of G has both incoming and outgoing edges. An *sT-digraph* is an acyclic digraph with exactly one source.

Let f be a face of planar drawing (or embedding) of a digraph. A *source-switch* (*sink-switch*) of f is a source (sink) of f. Note that a *source-switch* (*sink-switch*) is not necessarily a source (sink) of G.

An *upward* drawing of a digraph is such that all the edges are represented by directed curves increasing monotonically in the vertical direction. A digraph has an upward drawing if and only if it is acyclic. A digraph is *upward planar* if it admits a planar upward drawing. Note that a planar acyclic digraph does not necessarily have a planar upward drawing, as shown in Fig. 1b. An upward planar digraph also admits a planar upward straight-line drawing [21, 9]. A planar st-digraph is a planar digraph with exactly one source s and one sink t, connected by edge (s,t). A digraph is upward planar if and only if it is a subgraph of a planar st-digraph [21, 9].

A planar embedding of a digraph is *candidate* if the incoming (outgoing) edges around each vertex are consecutive. The planar embedding underlying an upward drawing is candidate.

An *upward embedding* of a digraph G is an embedding of G such that

- each source- and sink-switch of each face of G is labeled *small* or *large*;
- there exists a planar straight-line upward drawing of G where each switch labeled *small* corresponds to an angle with measure $< \pi$, and each switch labeled *large* has measure $> \pi$.

Finally, the following lemma is due to Hutton and Lubiw.

LEMMA 1 (see [19]). *If a digraph has a single source, then it is upward planar if and only if its biconnected components are upward planar.*

Due to this result, in the remainder of the paper we will consider only biconnected digraphs.

2.2. SPQR-trees. In the following we summarize SPQR-trees. For more details see [10, 11]. SPQR-trees are closely related to the classical decomposition of biconnected graphs into triconnected components [17].

Let G be a biconnected graph. A *split pair* of G is either a separation pair or a pair of adjacent vertices. A *split component* of a split pair $\{u,v\}$ is either an edge (u,v) or a maximal subgraph C of G such that C contains u and v, and $\{u,v\}$ is not a split pair of C. Note that a vertex w distinct from u and v belongs to exactly one split component of $\{u,v\}$.

Let $\{s,t\}$ be a split pair of G. A *maximal split pair* $\{u,v\}$ of G with respect to $\{s,t\}$ is a split pair of G distinct from $\{s,t\}$ such that, for any other split pair $\{u',v'\}$ of G, there exists a split component of $\{u',v'\}$ containing vertices u, v, s, and t.

Let $e(s,t)$ be an edge of G, called *reference edge*. The *SPQR-tree* $\mathcal{T}$ of G with respect to e describes a recursive decomposition of G induced by its split pairs. Tree $\mathcal{T}$ is a rooted ordered tree whose nodes are of four types: S, P, Q, and R. Each node μ of $\mathcal{T}$ has an associated biconnected multigraph, called the *skeleton* of μ, and denoted by *skeleton*(μ). Also, it is associated with an edge of the skeleton of the parent ν of μ, called the *virtual edge* of μ in *skeleton*(ν). Tree $\mathcal{T}$ is recursively defined as follows.

Trivial case. If G consists of exactly two parallel edges between s and t, then $\mathcal{T}$ consists of a single Q-node whose skeleton is G itself.

Parallel case. If the split pair $\{s,t\}$ has at least three split components $G_1,\ldots,G_k$ ($k \geq 3$), the root of $\mathcal{T}$ is a P-node μ. Graph *skeleton*(μ) consists of k parallel edges between s and t, denoted $e_1,\ldots,e_k$, with $e_1 = e$.

Series case. Otherwise, the split pair $\{s,t\}$ has exactly two split components, one of them is the reference edge e, and we denote with G' the other split component. If G' has cutvertices $c_1,\ldots,c_{k-1}$ ($k \geq 2$) that partition G into its blocks

$G_1, \ldots, G_k$, in this order from s to t, the root of $\mathcal{T}$ is an S-node μ. Graph *skeleton*(μ) is the cycle $e_0, e_1, \ldots, e_k$, where $e_0 = e$, $c_0 = s$, $c_k = t$, and e_i connects c_{i-1} with c_i $(i = 1, \ldots, k)$.

Rigid case. If none of the above cases applies, let $\{s_1, t_1\}, \ldots, \{s_k, t_k\}$ be the maximal split pairs of G with respect to $\{s, t\}$ $(k \geq 1)$, and for $i = 1, \ldots, k$, let G_i be the union of all the split components of $\{s_i, t_i\}$ except for the one containing the reference edge e. The root of $\mathcal{T}$ is an R-node μ. Graph *skeleton*(μ) is obtained from G by replacing each subgraph G_i with the edge e_i between s_i and t_i.

Except for the trivial case, μ has children $\mu_1, \ldots, \mu_k$ in this order such that μ_i is the root of the SPQR-tree of graph $G_i \cup e_i$ with respect to reference edge e_i $(i = 1, \ldots, k)$. Edge e_i is said to be the *virtual edge* of node μ_i in *skeleton*(μ) and of node μ in *skeleton*(μ_i). Graph G_i is called the *pertinent graph* of node μ_i, and of edge e_i.

The tree $\mathcal{T}$ so obtained has a Q-node associated with each edge of G, except the reference edge e. We complete the SPQR-tree by adding another Q-node, representing the reference edge e, and making it the parent of μ so that it becomes the root. Observe that we are defining SPQR-trees of graphs; however, the same definition can be applied to digraphs. An example of SPQR-tree is shown in Figure 2.

Let μ be a node of $\mathcal{T}$. We have the following:

- if μ is an R-node, then *skeleton*(μ) is a triconnected graph;
- if μ is an S-node, then *skeleton*(μ) is a cycle;
- if μ is a P-node, then *skeleton*(μ) is a triconnected multigraph consisting of a bundle of multiple edges;
- if μ is a Q-node, then *skeleton*(μ) is a biconnected multigraph consisting of two multiple edges.

The skeletons of the nodes of $\mathcal{T}$ are homeomorphic to subgraphs of G. The SPQR-trees of G, with respect to different reference edges, are isomorphic and are obtained one from the other by selecting a different Q-node as the root. Hence, we can define the *unrooted SPQR*-tree of G without ambiguity.

The SPQR-tree $\mathcal{T}$ of a graph G with n vertices and m edges has m Q-nodes and $O(n)$ S-, P-, and R-nodes. Also, the total number of vertices of the skeletons stored at the nodes of $\mathcal{T}$ is $O(n)$.

A graph G is planar if and only if the skeletons of all the nodes of the SPQR-tree $\mathcal{T}$ of G are planar. An SPQR-tree $\mathcal{T}$ rooted at a given Q-node represents all the planar drawings of G having the reference edge (associated with the Q-node at the root) on the external face (see Figure 2). Namely, such drawings can be constructed by the following recursive procedure:

- construct a drawing of the skeleton of the root ρ with the reference edge of the parent of ρ on the external face;
- for each child μ of ρ
  - let e be the virtual edge of μ in *skeleton*(ρ), and let H be the pertinent graph of μ plus edge e;
  - recursively draw H with the reference edge e on the external face;
  - in *skeleton*(ρ), replace virtual edge e with the above drawing of H minus edge e.

2.3. Upward planarity testing of embedded digraphs. In the remainder of this section we recall the combinatorial characterization of upward planarity for planar digraphs with a fixed embedding, given in [3, 4], which will be used extensively in this paper.

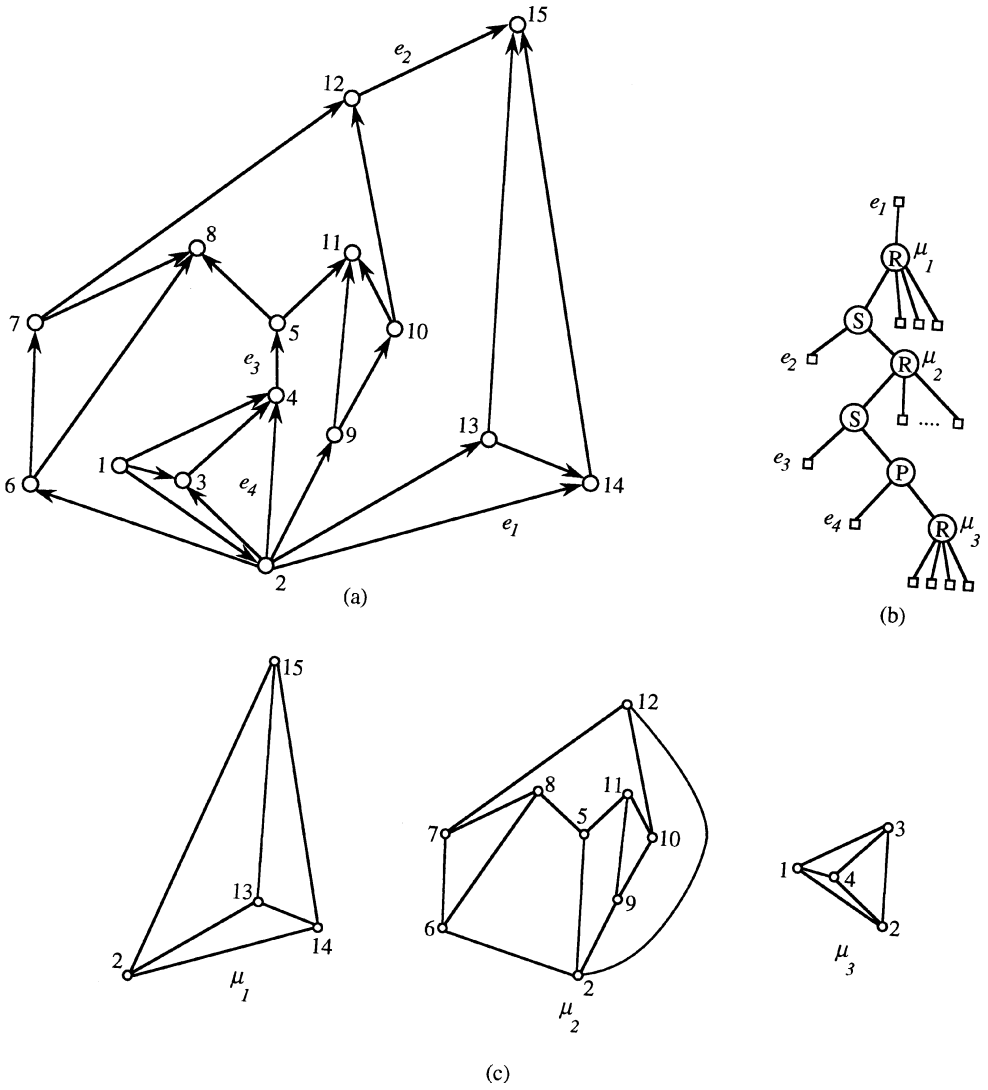


FIG. 2. (a) *A planar biconnected digraph G.* (b) *SPQR-tree $\mathcal{T}$ of G, where the* Q*-nodes are represented by squares.* (c) *Skeletons of the* R*-nodes.*

In [3, 4], the problem of testing whether an embedded planar digraph G admits a planar upward drawing is formulated as a perfect c-matching problem on a bipartite graph derived from G. To introduce this formulation we need some notation and definitions.

Let Γ be a planar straight-line upward drawing of an embedded upward planar digraph G. (As shown in [21, 9], every upward planar digraph admits a planar upward straight-line drawing.) We say that a sink t (source s) of G is *assigned* to a face f of Γ if the angle defined by the two edges of f incident on t (s) is greater than π. Informally speaking, t (s) is assigned to f if it "penetrates" into face f. Clearly, each sink (source) can be assigned only to one face, while an internal vertex is not assigned to any face. In [3, 4], it is shown that the number of vertices assigned to a face f in

any upward drawing is equal to the capacity $c(f)$ of the face itself, which is defined as follows. Let n_f be the number of sink-switches of f (n_f is also equal to the number of source-switches of f). We set $c(f) = n_f - 1$ if f is an internal face and $c(f) = n_f + 1$ if f is the external face. In the following, we associate to each vertex x and each face f the quantity $Z(x, f)$, where $Z(x, f) = 1$ if x is a switch of f, $Z(x, f) = 0$ otherwise. Clearly, we have that $2n_f = \sum_{x \in f} Z(x, f)$.

This intuitive idea of assignment of vertices to faces can be formally expressed as a perfect c-matching problem [24]. Namely, given a planar digraph G with a candidate planar embedding Ψ, we associate with G and Ψ the bipartite network $N(L_1, L_2, E_N)$ with vertex set $L_1 \cup L_2$ and edge-set E_N, where (i) the vertices of L_1 represent the sources and sinks of G; (ii) the vertices of L_2 represent the faces of Ψ; and (iii) E_N has an edge (v, f) if and only the vertex of G represented by $v \in L_1$ lies on the face of Ψ represented by $f \in L_2$. The c-matching problem for G and Ψ is described by the following equations:

$$\sum_{(v,f) \in E_N} x_{vf} = c(f), \quad \forall f \in L_2,$$

$$\sum_{(v,f) \in E_N} x_{vf} = 1, \quad \forall v \in L_1,$$

where $x_{vf} = 1$ indicates that vertex v is *assigned* to face f and $x_{vf} = 0$ indicates otherwise. A solution of this c-matching problem is called an *upward consistent assignment* of the variables x_{vf} and is denoted by $\mathcal{A}$. The equations of the first set are called *capacity equations*.

LEMMA 2 (see [3, 4]). *Let G be a digraph with a candidate planar embedding Ψ. Then Ψ is an upward embedding of G if and only if the c-matching problem associated with G and Ψ admits an upward consistent assignment.*

If $\mathcal{A}$ is an upward consistent assignment for Ψ, and f is a face of Ψ, we denote by $A(f)$ the set of vertices of G assigned to f in $\mathcal{A}$.

3. Embedded digraphs. In this section we give a new combinatorial characterization of upward planarity for planar single-source digraphs with a prescribed embedding. This characterization yields an optimal algorithm for testing whether an embedded planar single-source digraph has an upward planar drawing that preserves the embedding.

Given a planar single-source digraph G and an upward embedding Γ of G, from the first condition on the capacity equations of the perfect matching problem and from the fact that G has a unique source, the following properties can be easily derived (see Figure 3).

FACT 1. *The source of G is the bottommost vertex of Γ.*

FACT 2. *For the external face h of Γ, all the sink-switches are sinks of G and are assigned to h. (See Figure* 3a.*)*

FACT 3. *For each internal face f, at most one sink-switch (the topmost vertex of f in Γ) is not a sink of G and all but one sink switches are assigned to f. (See Figure* 3b.*)*

We shall also use the following result about cycles in planar single-source digraphs.

LEMMA 3. *Let G and G' be planar single-source digraphs such that G' is obtained from G by means of one of the following operations:*

- *adding a new vertex v and a new edge (u, v) or (v, u), connecting v to a vertex u of G;*

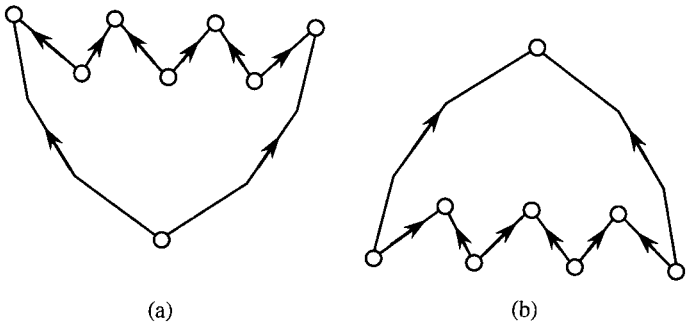


FIG. 3. *Schematic illustration of* (a) *the external face;* (b) *an internal face of an upward planar drawing of a planar single-source digraph.*

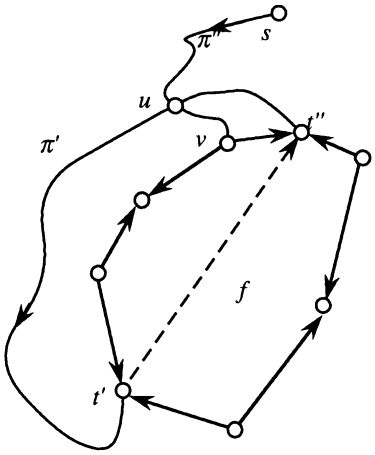


FIG. 4. *Illustration of the proof of Lemma* 3.

- *adding a directed edge between the source and a sink on the same face in some embedding of G;*
- *adding a directed edge between two sink-switches on the same face in some embedding of G.*

Then G is acyclic if and only if G' is acyclic.

Proof. The acyclicity is trivially preserved by the first two operations. Regarding the third operation, consider an embedding of G with the source on the external face, and assume, for a contradiction, that G is acyclic and that adding the edge (t', t'') between sink-switches t' and t'' of face f causes the resulting graph G' to have a cycle γ (see Figure 4). Cycle γ must consist of edge (t', t'') and a directed path π' in G from t'' to t'. Let v be the neighbor of t'' in f inside γ, and let π'' be a directed path from the source of G to v. Since the source is external to cycle γ, path π'' must have at least a vertex in common with path π'. Let u be the last vertex of π'' that is also on π'. We have that G has a cycle consisting of edge (v, t''), the subpath of π' from t'' to u, and the subpath of π'' from u to v, which is a contradiction. □

Given an embedded planar single-source digraph G, the *face-sink graph* F of G is the incidence graph of the faces and the sink-switches of G (see Figures 5a and 6).

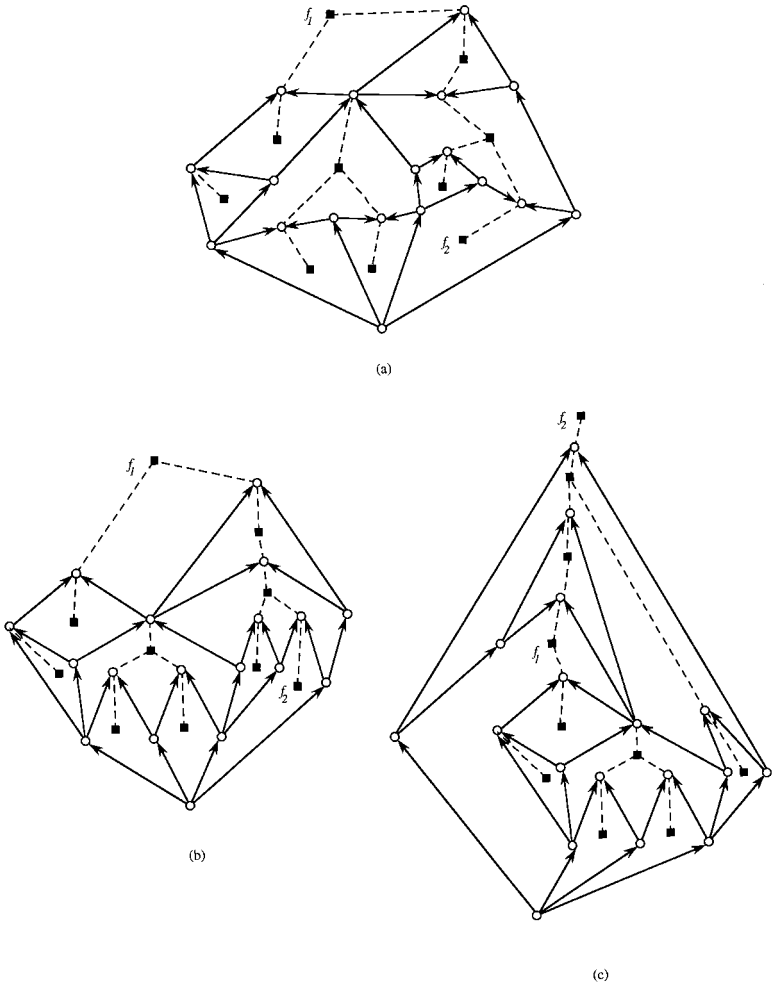


FIG. 5. (a) *An embedded planar single-source digraph G and its face-sink graph.* (b)–(c) *Upward drawings of G that preserve the embedding, with different external faces.*

Namely,

- the vertices of F are the faces and the sink-switches of G;
- graph F has an edge (f, v) if v is a sink-switch on face f.

THEOREM 1. *Let G be an embedded planar single-source digraph and let h be a face of G. Digraph G has an upward planar drawing that preserves the embedding with external face h if and only if all of the following conditions are satisfied:*

1. *graph F is a forest;*

2. *there is exactly one tree T of F with no internal vertices of G, while the remaining trees have exactly one internal vertex;*

3. *h is in tree T; and*

4. *the source of G is in the boundary of h.*

Also, if the above conditions are satisfied, then an embedded planar st-digraph G' containing G as an embedded subgraph is obtained as follows:

1. *root tree T at h and each remaining tree of F at its (unique) internal vertex;*

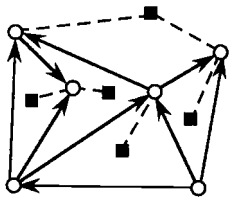


FIG. 6. *An embedded planar single-source digraph that does not have an upward drawing that preserves the embedding.*

2. *orient F by directing edges toward the roots;*
3. *prune the leaves from every tree of F; and*
4. *add the resulting forest $\hat{F}$ and the edge (s, h) to G.*

Proof. Only if. Let Γ be any planar upward drawing of G that preserves the original planar embedding and has external face h. By Fact 1, condition 4 is verified. Orient the face-sink graph F of G by directing edge (v, f) from v to f if v is a sink assigned to face f, and from f to v otherwise. By Facts 2–3, each vertex of F has at most one outgoing edge. Specifically, each internal face and each sink has exactly one outgoing edge, while each internal vertex and the external face have no outgoing edges. Now, label the vertices of F as follows: the label of a sink-switch is its y-coordinate in Γ; the label of an internal face f is $y(v) - \epsilon$, where v is the sink-switch not assigned to f, and ϵ is a suitably small positive real value, and the label of the external face h is $+\infty$. Since Γ is an upward drawing, the edges of F are directed by increasing labels. We conclude that F is a forest of sink-trees. One tree is rooted at h, while the other trees are rooted at internal vertices. This shows conditions 1–2. Condition 3 follows from Fact 1.

If. We show that, if F satisfies the conditions of the theorem, then G is a subgraph of a planar st-digraph G', which is obtained as the union of G and $\hat{F}$. This implies that G is upward planar. Planarity is preserved since a star is inserted in each face. Also, G' has exactly one source (s) and one sink (h) connected by a directed edge. It remains to be shown that G' is acyclic. By the construction of G' and Lemma 3, we have that G' is acyclic if and only if G is acyclic. Assume, for a contradiction, that G is not acyclic. Let γ be a cycle of G that does not enclose any other cycle. Note that the source s must be outside γ. If γ is a face of G, then F has an isolated vertex associated with face γ, which is a contradiction. Otherwise (γ is not a face of G), the subgraph $\hat{F}'$ of $\hat{F}$ enclosed by γ consists of a forest of trees, each with exactly one internal vertex. Let H be the digraph obtained from the subgraph of G enclosed by γ by removing the edges of γ, and adding a new vertex s' together with edges from s' to all the vertices of γ. By our choice of cycle γ, H is a planar single-source digraph. Adding $\hat{F}'$ to H yields a planar single-source digraph without sinks, and hence a digraph with cycles. By Lemma 3, H must also have cycles, which is again a contradiction. □

Theorem 1 is illustrated in Figures 5–6. The following algorithm tests whether an embedded planar single-source digraph G is upward planar, and reports all the faces of G that can be external in an upward planar drawing of G with the prescribed embedding.

ALGORITHM. *Embedded-Test.*

1. Construct the face-sink graph F of G.
2. Check conditions 1 and 2 of Theorem 1. If these conditions are not verified,

then return "not-upward-planar" and stop.

3. Report the set of faces of G that contain vertex s in their boundaries and are associated with nodes of tree T. If such a set of faces is empty, then return "not-upward-planar"; else return "upward-planar."

For the example of Figure 5, Algorithm *Embedded-Test* returns "upward-planar" and reports two faces.

THEOREM 2. *Let G be an embedded planar single-source digraph with n vertices. Algorithm* embedded test *determines whether G has an upward planar drawing that preserves the embedding and reports all the admissible external faces. It runs in $O(n)$ sequential time and in $O(\log n)$ time on a CRCW PRAM with $n \cdot \alpha(n)/\log n$ processors, using $O(n)$ space.*

Proof. The correctness of the algorithm follows directly from Theorem 1. All the steps can be performed sequentially in $O(n)$ time with straightforward methods.

Regarding the parallel complexity, steps 1 and 3 take $O(\log n)$ time on a CREW PRAM with $n/\log n$ processors, using list-ranking [5]. Step 2 can be executed by computing a spanning forest of the face-sink graph, which takes $O(\log n)$ time on a CRCW PRAM with $n \cdot \alpha(n)/\log n$ processors [5], and thus determines the parallel time complexity. □

4. Upward planarity and SPQR-trees. Let G be a biconnected single-source digraph. In this section we give a combinatorial characterization of the upward planarity of G using SPQR-trees.

4.1. Basic definitions and main result. A digraph is *expanded* if each internal vertex has exactly one incoming edge or one outgoing edge. The *expansion* of a digraph is obtained by replacing each internal vertex v with two new vertices v_1 and v_2, which inherit the incoming and outgoing edges of v, respectively, and the edge (v_1, v_2). Observe that a digraph is acyclic if and only if its expansion is acyclic. A planar embedding of an expanded digraph is candidate. In the remainder of this section we consider only expanded digraphs because of the following property.

FACT 4. *A digraph is upward planar if and only if its expansion is upward planar.*

Let $G' = (V', E')$ and $G'' = (V'', E'')$ be two digraphs. The *union* of G' and G'', denoted by $G' \cup G''$, is a digraph $G = (V, E)$ with $V = V' \cup V''$ and $E = E' \cup E''$, i.e., G is obtained from G' and G'' by identifying the vertices in V' and V'' with common labels.

Let $\{u, v\}$ be a separation pair of G that decomposes G into p split components $J_1, \ldots, J_p$. We call a *component separated by* $\{u, v\}$ or simply a *component* any digraph obtained as the union of q of the split components in $\{J_1, \ldots, J_p\}$, with $0 < q < p$.

We call *peak* a digraph consisting of three vertices a, b, and t, and two directed edges (a, t) and (b, t). See Figure 7b.

Let K be a component of G with respect to the separation pair $\{u, v\}$. In the following, we denote with $G - K$ the digraph obtained from G by deleting every vertex belonging to K, except for the vertices u and v, and by K° the digraph obtained from K by deleting vertices u and v. Also, for simplicity we write $G - K_1 - K_2 - \cdots - K_m$ instead of $(\cdots((G - K_1) - K_2) \cdots - K_m)$.

Finally, we associate with a component K of G either a directed edge or a peak (see Figures 7a–b) according to the following rules.

Rule 1. u and v are sources of K: a peak with $a \equiv u$ and $b \equiv v$.

Rule 2. u is a source of K and v is a sink of K.

(a) $s \notin K^\circ$: a directed edge (u, v).

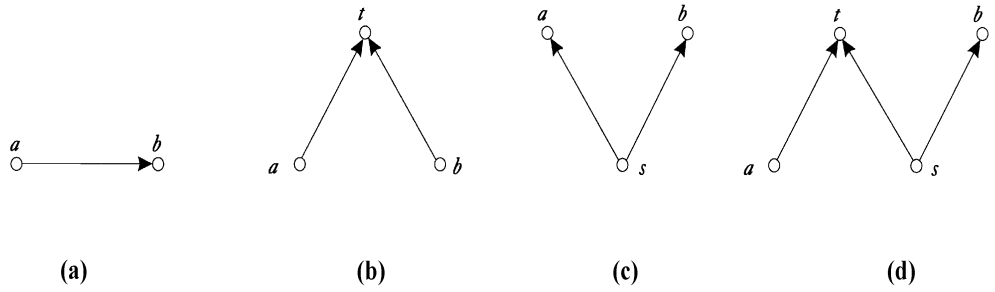


FIG. 7. (a) *Directed edge;* (b) *peak;* (c) *valley;* (d) *zig-zag.*

(b) $s \in K^\circ$: a peak with $a \equiv u$ and $b \equiv v$.

Rule 3. u is a source of K and v is an internal vertex of K.

(a) v is a source of $G - K$ and $s \notin K^\circ$: a directed edge (u, v).

(b) v is not a source of $G - K$ or $s \in K^\circ$: a peak with $a \equiv u$ and $b \equiv v$.

Rule 4. u and v are not sources of K.

(a) u is a source of $G - K$: a directed edge (u, v).

(b) u is not a source of $G - K$: a directed edge (v, u).

The digraph associated to K by the above rules is called *directed-virtual-edge* and will be denoted by $d(K, G - K)$. Observe that the choice of the directed-virtual-edge depends, in general, both on K and on $G - K$.

We call *minor* of G either G itself or the digraph

$$G - K_1 - \cdots - K_m \cup d(K_1, G - K_1) \cup \cdots \cup d(K_m, G - K_m),$$

where $K_1, \ldots, K_m, m \geq 1$, are components of G with the property that no two components share a common edge. In other words, the digraph $G - K_1 - \cdots - K_m \cup d(K_1, G - K_1) \cup \cdots \cup d(K_m, G - K_m)$ is obtained from G by replacing $K_1, \ldots, K_m$ with the corresponding directed-virtual-edges (see Figure 8). Observe that, in general, a minor of a minor of G is not a minor of G.

Let G be a planar single-source digraph, and let $\mathcal{T}$ be its SPQR-unrooted tree. The *sT-skeleton* of a node μ of $\mathcal{T}$, denoted by sT-skeleton(μ), is the minor of G obtained from the skeleton of μ by replacing each virtual edge with the directed-virtual-edge associated to its pertinent digraph. The *reference directed-virtual-edge* is the directed-virtual-edge associated with the pertinent digraph of the reference edge of μ. Examples of sT-skeletons are shown in Figure 9.

The main result of this section is summarized in the following theorem.

THEOREM 3. *A biconnected acyclic single-source digraph G is upward planar if and only if there exists a rooting of the SPQR-tree $\mathcal{T}$ of the expansion of G at a reference edge containing the source, such that the sT-skeleton of each node μ of $\mathcal{T}$ has a planar upward drawing with the reference directed-virtual-edge on the external face.*

The proof of Theorem 3 is given in the next two sections. Section 5 shows the *only-if* part, while section 6 shows the *if* part. Here we give some preliminary lemmas that will be used in the next sections.

4.2. Preliminary lemmas. In the following, S_G, T_G, and I_G will denote the set of sources, sinks, and internal vertices of G, respectively. If G has exactly one source, we denote such source by $s(G)$ (or simply by s, when no confusion arises).

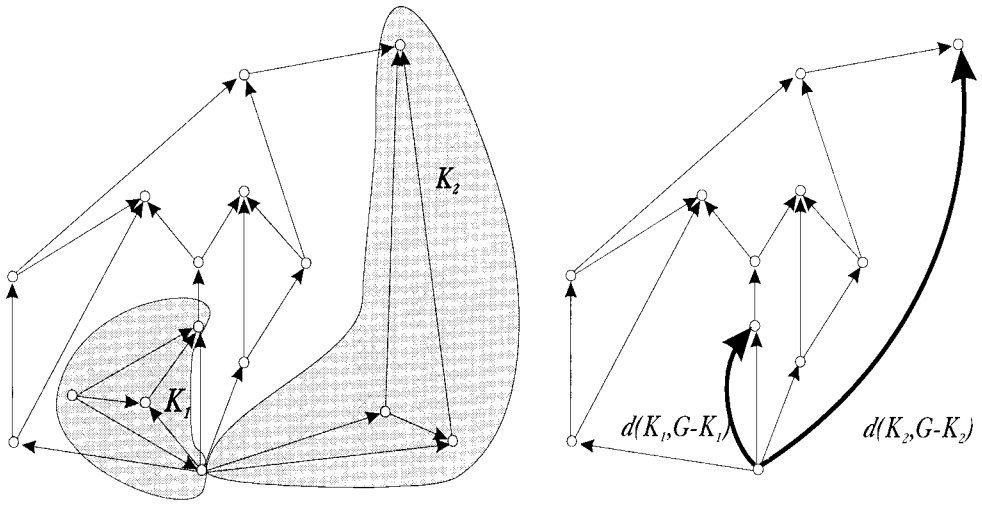


FIG. 8. *Construction of a minor.*

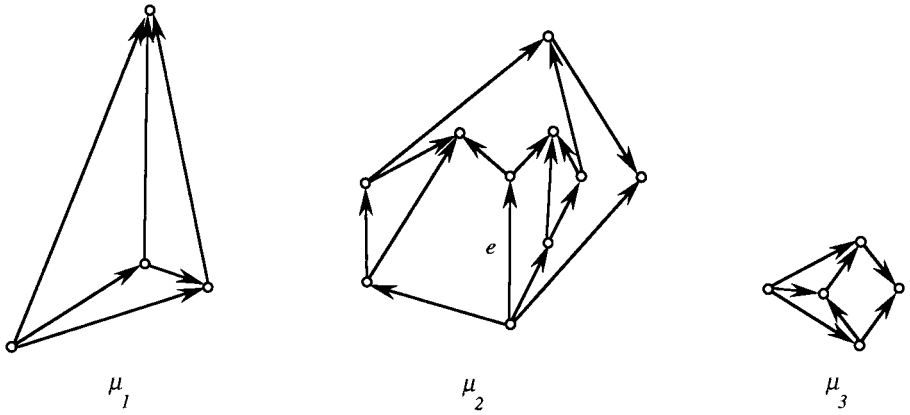


FIG. 9. *The sT-skeletons of the* R*-nodes of the digraph of Figure* 2.

We will make use of the following operations defined on an edge $e = (x, y)$ of a digraph G:

- *Contraction* (denoted by G/e) transforms G into a digraph G' obtained from G by removing edge e and by identifying vertices x and y.
- *Direct subdivision* transforms G into a digraph G' obtained from G by removing edge e and by adding a vertex z and edges (x, z) and (z, y).

We say that digraphs G_1 and G_2 are *homeomorphic* if both can be obtained by performing a finite number of direct subdivisions of a digraph G. Observe that we can have $G_1 = G$ or $G_2 = G$.

We call *valley* a digraph consisting of three vertices s', a, and b, and two directed edges (s', a) and (s', b) (see Figure 7c). Also, we call *zig-zag* a digraph consisting of four vertices s', t, a, and b, and three directed edges (s', t), (s', b), and (a, t) (see Figure 7d).

We denote by $x \to y$ a path from vertex x to vertex y. A vertex u is said to be *dominated* by vertex v if there is a path $v \to u$. We say that vertices x and y are

incomparable in G, denoted by $x \parallel y$, if there exists in G neither a path $x \to y$ nor a path $x \to y$.

FACT 5. *Let G be an sT-digraph. Then every vertex of G is dominated by s.*

FACT 6. *Let G be an acyclic digraph. G contains a source.*

LEMMA 4 (see [19]). *If G is an sT-digraph with $u \parallel v$ in G, then there exists in G a subgraph homeomorphic to a valley, with $a \equiv u$ and $b \equiv v$.*

LEMMA 5 (see [19]). *Let G be a connected acyclic digraph with exactly two sources u and v. Then there exists in G a subgraph homeomorphic to a peak, with $a \equiv u$ and $b \equiv v$.*

FACT 7. *Let G be a biconnected digraph and let $\{u, v\}$ be a separation pair of G. Neither u nor v is a cut-vertex of any component of G with respect to $\{u, v\}$.*

LEMMA 6. *Let G be an sT-digraph and let $\{u, v\}$ be a separation pair of G. Let K be a component with respect to $\{u, v\}$ such that $v \in I_K$ and K has exactly one source u. Then digraph K contains a subgraph homeomorphic to a peak, with $a \equiv u$ and $b \equiv v$.*

Proof. Let w be a vertex of K such that there exist two vertex disjoint directed paths $u \to w$ and $v \to w$ contained in K. The subgraph of K consisting of all edges and vertices belonging to the two paths is homeomorphic to a peak. Suppose w does not exist. Let V_v be the set of all vertices dominated by v in K (except v).

Let $\bar{V}_v$ be the set of all vertices not dominated by v in K. Observe that both V_v and $\bar{V}_v$ are nonempty and disjoint. Also, $V_v \cup \bar{V}_v \cup \{v\}$ is the set of vertices of K. Since v is not a cut-vertex of K, there is an edge connecting a vertex x of V_v with a vertex y of $\bar{V}_v$. If such an edge is (x, y), then y belongs to V_v, which is a contradiction. If such an edge is (y, x), then x is dominated by both v and u, and $x = w$, which is a contradiction. □

Finally, the following fact and lemmas concerning the embeddings of G will be used in the proof of the main theorem.

FACT 8. *Let G be a digraph and let G' be a digraph homeomorphic to a subgraph G'' of G. We have that*

(i) *if G is acyclic, then G' is acyclic.*
(ii) *if G is expanded, then G' is expanded.*
(iii) *if G is upward planar, then G' is upward planar.*
(iv) *if G'' is an sT-digraph, then G' is an sT-digraph.*

LEMMA 7 (see [19]). *Let G be a digraph, let (u, v) be an edge of G, and let vertex $u(v)$ have out-degree (in-degree) 1 in G. Let $G' = G/(u, v)$. We have that*

(i) *if G is acyclic then G' is acyclic.*
(ii) *if G is upward planar, then G' is upward planar.*

LEMMA 8. *Let G be an upward planar sT-digraph, and let Ψ_G be an upward embedding of G. Let $e = (s, u)$ be an edge of G embedded on the external face α of Ψ_G. Let $G' = G - e \cup P$ where P is a valley, i.e., P is the path $\{(s', s), (s', u)\}$. Then G' is an sT-digraph and has an upward embedding $\Psi_{G'}$, with P embedded on the external face.*

Proof. From Ψ_G we simply derive a candidate planar embedding $\Psi_{G'}$ of G' by replacing the edge (s, u) of G with the path P (see Figure 10).

We now show that $\Psi_{G'}$ is an upward embedding. This is done by deriving an upward consistent assignment $\mathcal{A}'$ associated to $\Psi_{G'}$ from the upward consistent assignment $\mathcal{A}$ associated to Ψ_G.

Let β be the internal face of Ψ_G containing the edge (s, u). We denote by $\gamma_1, \ldots, \gamma_p$ the faces of Ψ_G different from α and β.

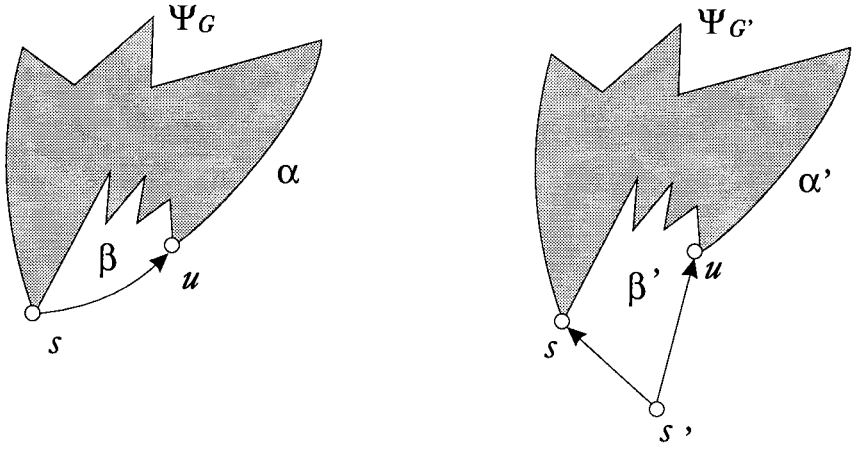


FIG. 10. *Construction of* $\Psi_{G'}$.

Clearly, there is a one-to-one correspondence between the faces of Ψ_G and the faces of $\Psi_{G'}$. We denote by α', β', $\gamma'_1, \ldots, \gamma'_p$ the faces of $\Psi_{G'}$ corresponding to $\alpha, \beta, \gamma_1, \ldots, \gamma_p$.

Also, it is $\gamma'_i = \gamma_i$, for $i = 1, \ldots, p$, and thus $c(\gamma'_i) = c(\gamma_i)$, for $i = 1, \ldots, p$. We show that $c(\alpha') = c(\alpha)$ and $c(\beta') = c(\beta)$.

Following the notation introduced in section 2.3, we have that $2n_{\alpha'} = 2n_\alpha - Z(s, \alpha) + Z(s, \alpha') - Z(u, \alpha) + Z(u, \alpha') + Z(s', \alpha')$. Since the edge (s, u) of α has been replaced by the edge (s', u) of α', it is $Z(u, \alpha') = Z(u, \alpha)$. It is easy to see that $Z(s, \alpha) = 1$, $Z(s, \alpha') = 0$, and $Z(s', \alpha') = 1$. Thus, $2n_{\alpha'} = 2n_\alpha$ and then $c(\alpha') = c(\alpha)$. In the same fashion, we can prove that $c(\beta') = c(\beta)$.

Let S (S') and T (T') be the set of sources and sinks of G (G'), respectively. Clearly $S' = S - \{s\} \cup \{s'\}$, and $T' = T$. Observe that, by Theorem 1, $s \in A(\alpha)$. $\mathcal{A}'$ is derived from $\mathcal{A}$ in the following way:

- $A'(\gamma'_i) = A(\gamma_i)$ for $i = 1, \ldots, p$;
- $A'(\alpha') = A(\alpha) - \{s\} \cup \{s'\}$; and
- $A'(\beta') = A(\beta)$.

It is straightforward to prove that $|A'(f)| = c(f)$ for each face $f \in \Psi_{G'}$. Since $\Psi_{G'}$ is a candidate embedding and $\mathcal{A}'$ is upward consistent, by Lemma 2, G' is upward planar. □

5. Proof of necessity for Theorem 3.

LEMMA 9. *Let G be a planar expanded sT-digraph G, $\{u, v\}$ a separation pair of G, and K a component with respect to $\{u, v\}$. Let $H = G - K$ and let $d_K = d(K, H)$ be the directed-virtual-edge associated to K with respect to G. Finally, let H' be the minor $H \cup d_k$. We have that*

(i) *H' is an expanded, acyclic sT-digraph.*

(ii) *if G is upward planar, then H' is upward planar.*

(iii) *if G is upward planar and $s(G) \in K^\circ$, then H' has an upward embedding with d_k on the external face.*

Proof. The following cases are possible, each corresponding to one of Rules 1–4 (in each case, the proof of (iii) is trivial and thus omitted).

1. u and v are sources of K.

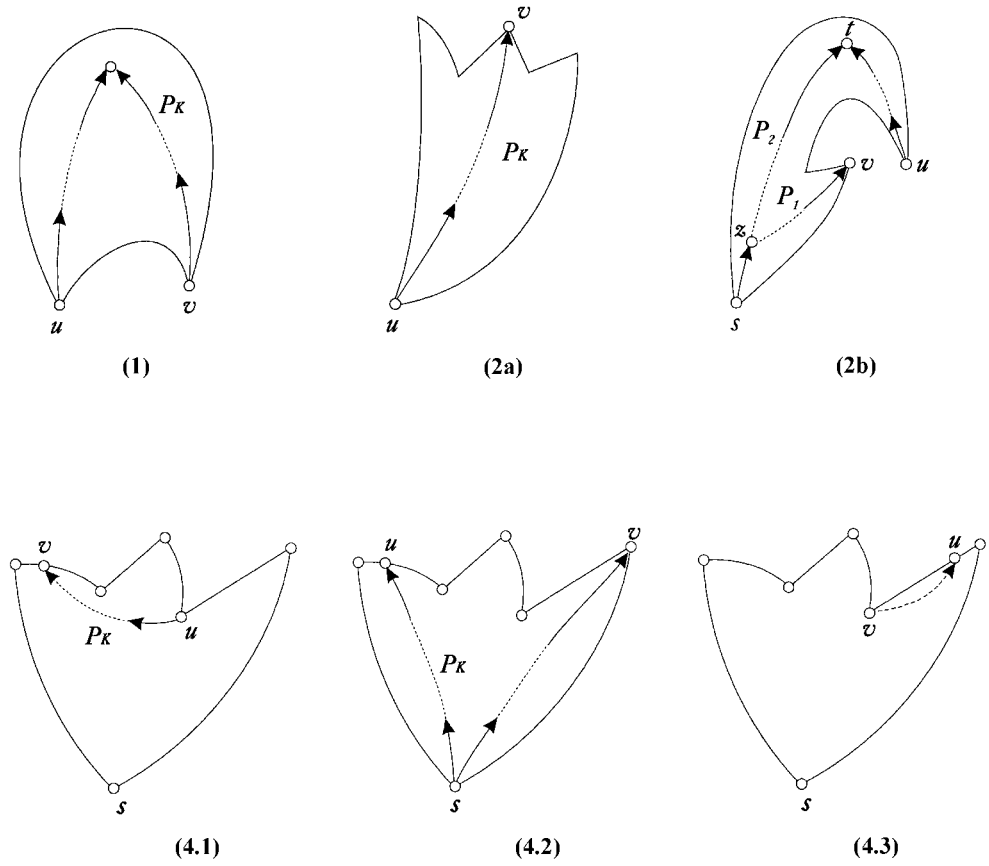


FIG. 11. *Various cases in the proof of Lemma* 9.

By Lemma 5, there is in K a path P_K homeomorphic to a peak, with $a \equiv u$ and $b \equiv v$ (see Figure 11). Thus, $H' = H \cup d_K$ is homeomorphic to a subgraph of G.

(i) By Fact 8, H' is an expanded acyclic digraph. We show now that it contains only one source. Vertices u and v are not both sources in H; otherwise G would contain two sources. Suppose one of u, v (say, u) is a source of H. Then $u \equiv s(G)$ and $u \equiv s(H')$; thus u is the only source of H'. Suppose neither u nor v is a source in H; then $s(G) \in H^\circ$ and s is the only source of H'.

(ii) By Fact 8, H' is upward planar.

2(a) u is a source of K and v is a sink of K and $s \notin K^\circ$.

Since $s \notin K^\circ$, u is the only source of K and there exists in K a path $u \to v$. Thus there is in K a path P_K homeomorphic to the directed edge (u, v), and then there is in G a subgraph homeomorphic to $H' = H \cup d_K$ (see Figure 11).

(i) By Fact 8, H' is an acyclic expanded digraph. Because of the existence of edge (u, v), v is not a source of H'. If $u \in S_H$, then $s(G) \equiv u$; thus u is the only source of H'. If $u \notin S_H$, $s(G) \in H^\circ$, and $s(G) \equiv s(H')$.

(ii) By Fact 8, H' is upward planar.

2(b) u is a source of K and v is a sink of K and $s \in K^\circ$.

Since $s \in K^\circ$, $s \neq u$ and, by Lemma 5, there is a vertex t distinct from s and u,

and two vertex disjoint paths $s \to t$ and $u \to t$. Note that u is not a source of H; otherwise u is a source of G, which is a contradiction. So v is the only source of H and since there is in G a path $s \to v$, we have that there is a path $s \to v$ in K. Moreover, since v is the only source of H, there is a path $v \to u$ in H and thus there is a path $v \to u$ in G. This implies that the path $s \to v$ and the path $u \to t$ are node-disjoint; otherwise there would be a path $u \to v$ and G would contain a cycle, which is a contradiction.

Let P_1 and P_2 be a path $s \to v$ and a path $s \to t$, respectively. Let z be the last vertex common to paths P_1 and P_2. Note that $z \neq t$, since P_1 and any path $u \to t$ are disjoint. Thus there exists in K a path P_K homeomorphic to a zig-zag, and then there is in G a subgraph homeomorphic to $H'' \equiv H \cup (z, v) \cup (z, t) \cup (u, t)$ (see Figure 11).

Observe that in H'', vertex v has in-degree 1. If we contract the edge (z, v) we obtain $H''/(z, v) \equiv H \cup d_K = H'$.

(i) By Fact 8 H'' is acyclic and expanded; thus, by Lemma 7 H' is acyclic. Note that every vertex except v has the same in- and out-degrees in H'' and H'. Since H'' is expanded and v is a source of H', H' is an expanded digraph. Moreover, since v is the only source of H, v is the only source of H'.

(ii) By Fact 8 H'' is upward planar; thus, by Lemma 7 H' is upward planar.

3(a) u is a source of K and v is an internal vertex of K; v is a source of $G - K$ and $s \notin K^\circ$.

Since u is the only source of K, there is a path $u \to v$ in K and the proof is analogous to that of case 2(a).

3(b) u is a source of K and v is an internal vertex of K; v is not a source of $G - K$ or $s \in K^\circ$.

1. If $s \in K^\circ$, then $s \neq u$ and K contains two sources; the proof is as in case 2(b).

2. If $s \notin K^\circ$, then u is the only source of K and, by Lemma 6, K contains a path P_K homeomorphic to a peak, with $u \equiv a$ and $v \equiv b$. Then $H' = H \cup d_K$ is homeomorphic to a subgraph of G.

(i) By Fact 8 H' is an expanded, acyclic digraph. If $u = s(G)$, then vertex u is the only source of H'. If $u \neq s(G)$, then u is not a source of H (otherwise G contains two sources, which is a contradiction), and $s(G)$ is the only source of H'. Thus, H' is an sT-digraph.

(ii) By Fact 8 H' is upward planar.

4. u and v are not sources of K.

Since u and v are not sources of K, we have that $s(G) \in K^\circ$. Then either u or v (or both) is a source of H. We discuss only the case where u is a source of H and Rule 4(a) is applied. In fact, if u is not a source of H, then v is a source of H and Rule 4(b) is applied; that is, the roles of u and v are interchanged. Three cases are possible.

1. If there exists a path $u \to v$ in K, then there is in K a path P_K homeomorphic to the directed edge (u, v) (see Figure 11).

(i) By Fact 8 H' is an expanded, acyclic digraph. Because of the existence of the edge (u, v), u is the only source of H'.

(ii) By Fact 8 H' is upward planar.

2. If u and v are incomparable in K, by Lemma 4 there exists in K a path P_K homeomorphic to a valley with $a \equiv u$ and $b \equiv v$ (see Figure 11). Thus, $H'' \equiv H \cup (s, v) \cup (s, u)$ is homeomorphic to a subgraph of G. Observe that in H'', vertex u has in-degree 1. If we contract the edge (s, u) we obtain $H''/(s, u) \equiv H \cup d_K = H'$.

(i) By Fact 8 H'' is acyclic and expanded; thus by Lemma 7 H' is acyclic. Note that all vertices except u have the same in- and out-degrees in H'' and H'. Since H'' is expanded and u is a source of H', then H' is an expanded digraph. Moreover, since $s(G) \in K^\circ$, u is the only source of H'.

(ii) By Fact 8 H'' is upward planar; thus, by Lemma 7 H' is upward planar.

3. If there exists a path P_K from v to u in K, then v is also a source of H (see Figure 11). Otherwise u would be the only source of H, so there would be a path $u \to v$ in H and thus a path $u \to v$ in G, which is a contradiction. So both u and v are sources of H.

(i) Since H is an expanded, acyclic digraph, and since both vertices u and v are sources of H, $H' = H \cup (u, v)$ is expanded and acyclic. Furthermore, it contains only one source u.

(ii) Let H'' be the digraph $H \cup (v, u)$. Digraph H'' is homeomorphic to a subgraph of G and thus, by Fact 8, there exists an upward embedding $\Psi_{H''}$ of H'', with (v, u) on the external face. Furthermore, v is the only source of H'', and H'' is an expanded sT-digraph.

H' can be obtained from H'' by the means of the following operations:

1. Construct $\bar{H}$ from H'' by adding a vertex s' and replacing the edge (v, u) with the valley $\{(s', v), (s', u)\}$.

2. Construct H' from $\bar{H}$ by contracting the edge (s', u).

By Lemma 8, $\bar{H}$ has an upward embedding with the valley $\{(s', v), (s', u)\}$ on the external face. By Lemma 7, H' has an upward embedding with the edge (u, v) on the external face. □

Following the developments of the previous proof, for each case a particular path P_K is detected in the component K. These paths and the corresponding cases will play a central role in the next section.

The previous lemma refers to a particular minor of G obtained by replacing exactly one component, but it can be easily extended to any minor of G. Before proving the next lemma, we must observe a property of sources in minors. Suppose u is a source of G. Then u is a source in every component of G that contains u. By a simple inspection of Rules 1–4 we have the following.

FACT 9. *Let G be a digraph and K be a component of G with respect to the separation pair $\{u, v\}$. Let G' be a minor of G such that $K \subset G'$. If u is a source of $G - K$, then u is a source in $G' - K$.*

LEMMA 10. *Let G be an expanded sT-digraph and let $\tilde{H}' = G - K_1 - \cdots - K_m \cup d_{K_1} \cdots \cup d_{K_m}$ be a minor of G, where $d_{K_i} = d(K_i, G - K_i)$. Then*

(i) *$\tilde{H}'$ is an expanded sT-digraph.*

(ii) *if G is upward planar, then $\tilde{H}'$ is upward planar.*

(iii) *if G is upward planar and $s(G) \in K_i^\circ$, $1 \le i \le m$, then H' has an upward embedding with d_{K_i} on the external face.*

Proof. The proof is by induction. When $m = 1$, the minor is $\tilde{H}' = G - K_1 \cup d(K_1, G - K_1)$, and by Lemma 9 the basis of the induction holds. Now, suppose that the minor $\tilde{G} = G - K_1 - \cdots - K_{l-1} \cup d_{K_1} \cup \cdots \cup d_{K_{l-1}}$ is an expanded sT-digraph and is upward planar. We show that the minor $\tilde{J}' = G - K_1 - \cdots - K_l \cup d_{K_1} \cup \cdots \cup d_{K_l}$ is an expanded sT-digraph and is upward planar.

Note first that $\tilde{J}' = \tilde{G} - K_l \cup d_{K_l}$. If we can prove that $d_{K_l} = d(K_l, G - K_l)$ is equal to $d(K_l, \tilde{G} - K_l)$, then the thesis follows by Lemma 9. In other words, we have to prove that the directed-virtual-edge which substitutes K_l remains the same when Rules 1–4 are applied to the pair $(K_l, \tilde{G} - K_l)$ rather than to the pair $(K_l, G - K_l)$.

When Rules 1 and 2 are applied, the directed-virtual-edge depends only on the component K_l, and $d_K = d(K_l, G - K_l) = d(K_l, \tilde{G} - K_l)$. Hence we have to consider only Rules 3 and 4.

3(a) u is a source of K_l and v is an internal vertex of K_l, v is a source of $G - K_l$ and $s(G) \notin K_l^\circ$.

By Fact 9, v is a source in $\tilde{G} - K_l$. Moreover, since $s(G) \notin K_l^\circ$ we have that $s(\tilde{G}) \notin K_l^\circ$, and Rule 3(a) must be applied to the pair $(K_l, \tilde{G} - K_l)$.

3(b) u is a source of K_l and v is an internal vertex of K_l; v is not a source of $G - K_l$ or $s(G) \in K_l^\circ$.

If $s \in K_l^\circ$, Rule 3(b) must be applied to the pair $(K_l, \tilde{G} - K_l)$.

Suppose that v is not a source of $G - K_l$ and $s \notin K_l^\circ$. Note that v is a sink in $G - K_l$ (v cannot be internal in $G - K_l$ since v is internal in K_l and G is expanded).

If v is not a source of $\tilde{G} - K_l$ then Rule 3(b) must be applied to the pair $(K_l, \tilde{G} - K_l)$.

Suppose v is a source of $\tilde{G} - K_l$; then there exists a component K_q of $G - K_l$, with $0 < q < l$, with $K_q \notin \tilde{G} - K_l$, having v and u_q as poles, whose directed-virtual-edge is either a peak or the edge (v, u_q). Observe that v is a sink of K_q. If $s \notin K_q^\circ$ then u_q is the only source of K_q. Hence Rule 2(a) is applied to the pair $(K_q, G - K_q)$ and $d(K_q, G - K_q)$ is the edge (u_q, v), which is a contradiction.

If $s \in K_q^\circ$ then u_q is a source of $G - K_q$ (since v is internal in $K_l \in G - K_q$). Thus u_q is not a source of K_q; otherwise G contains two sources. Then Rule 4(a) is applied to the pair $(K_q, G - K_q)$ and $d(K_q, G - K_q)$ is the edge (u_q, v), which is a contradiction.

4(a) u and v are not sources of K_l; u is a source of $G - K_l$. By Fact 9, u is a source of $\tilde{G} - K_l$ and Rule 4(a) can be applied to the pair $(K_l, \tilde{G} - K_l)$.

4(b) u and v are not sources of K_l; u is a not a source of $G - K_l$. Since u is not a source in $G - K_l$ and $s(G) \in K_l$, then v is a source in $G - K_l$ and v is a source in $\tilde{G} - K_l$. So Rule 4(a) can be applied to the pair $(K_l, \tilde{G} - K_l)$, with v and u interchanged. □

The proof of the necessity of Theorem 3 is now a simple corollary of Lemma 10. In fact, for each node μ of tree $\mathcal{T}$, the sT-skeleton of μ is a minor of G.

6. Proof of sufficiency for Theorem 3.

LEMMA 11. *Let G be a planar expanded sT-digraph, $\{u, v\}$ a separation pair of G, and K a component with respect to $\{u, v\}$ such that $s(G) \in K$. Let $H = G - K$ and let $d_K = d(K, H)$ and $d_H = d(H, K)$ be the directed-virtual-edges associated to K and H (with respect to G), respectively. Finally, let H' be the minor $H \cup d_K$ and K' be the minor $K \cup d_H$. If K' is upward planar and H' has an upward embedding with d_K on the external face, then G is upward planar.*

Before proving the above lemma, we need some preliminary results, namely, the following Lemmas 12–16.

We remind the reader that in the proof of Lemma 9, in correspondence with each case of the proof, a path P_K is detected in the component K. Such a path P_K will be used in the following lemma (see Figure 12).

LEMMA 12. *Let G be a planar expanded sT-digraph G, $\{u, v\}$ a separation pair of G, and K a component with respect to $\{u, v\}$. Let $H = G - K$ and let $d_K = d(K, H)$ be the directed-virtual-edge associated to K with respect to G. Let H' be the minor $H \cup d_K$ and let $\bar{H} = H \cup P_K$. Suppose H' has an upward embedding $\Psi_{H'}$, with d_K embedded on the external face if $s(G) \in K^\circ$. Denote by $\Psi_H \subset \Psi_{H'}$ the upward embedding of H contained in $\Psi_{H'}$ and let α_H be the face of Ψ_H in which d_K is embedded. We have that*

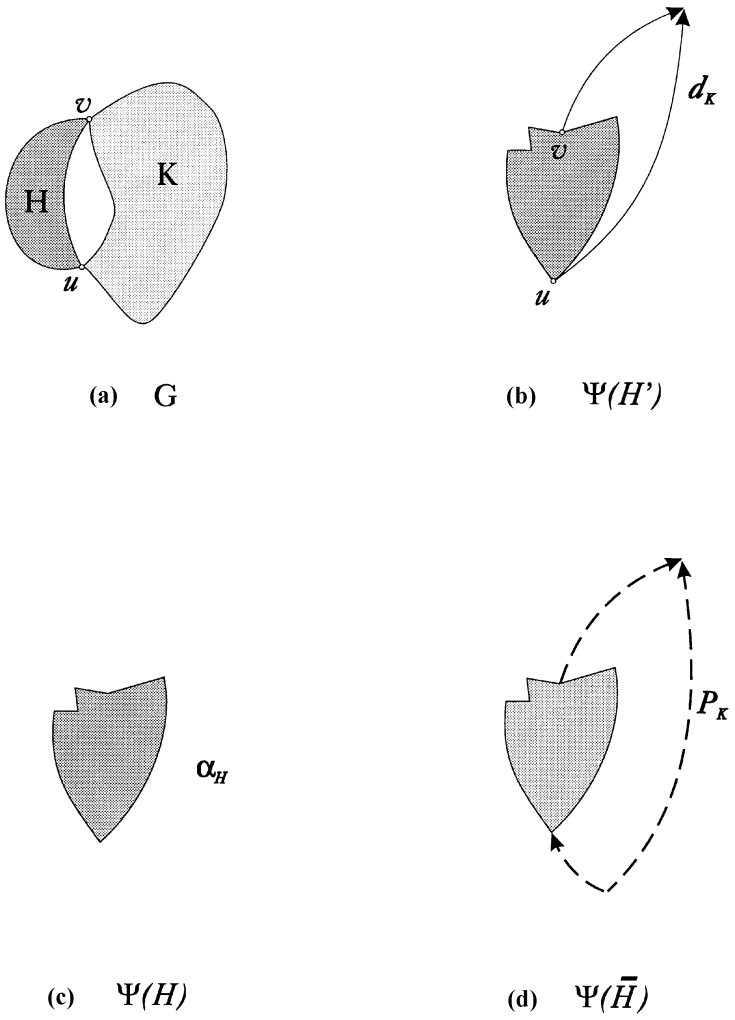


FIG. 12. *Illustration of the statement of Lemma* 12.

(i) $\bar{H} = H \cup P_K$ *is an expanded, acyclic digraph.*

(ii) $\bar{H}$ *is an sT-digraph and has an upward embedding* $\Psi_{\bar{H}}$*, with* $\Psi_H \subset \Psi_{\bar{H}}$ *and* P_K *embedded in face* α_H *of* Ψ_H*.*

Proof. Since $\bar{H}$ is a subgraph of G then it is expanded and acyclic and (i) holds.

We now prove (ii). Since both d_K and P_K depend on the component K, we distinguish the following four cases, each corresponding to the cases of the proof of Lemma 9.

1. P_K is homeomorphic to d_K; hence $\bar{H}$ is homeomorphic to H'. By Fact 8, the lemma holds.

2.(a) P_K is homeomorphic to d_K; hence $\bar{H}$ is homeomorphic to H'. By Fact 8, the lemma holds.

(b) Since $s \in K^\circ$, d_K is embedded on the external face of $\Psi_{H'}$. Observe that $\bar{H}$ can be obtained from H' substituting the edge (v, t) with a path homeomorphic to a valley. Thus, by Lemma 8 and by Fact 8, the lemma holds.

3.(a) P_K is homeomorphic to d_K; hence $\bar{H}$ is homeomorphic to H'. By Fact 8, the lemma holds.

(b) If $s \in K^\circ$ then the proof is as in case 2(b). If $s \notin K^\circ$ then P_K is homeomorphic to d_K; hence $\bar{H}$ is homeomorphic to H'. By Fact 8, the lemma holds.

4. (1) P_K is homeomorphic to d_K; hence $\bar{H}$ is homeomorphic to H'. By Fact 8, the lemma holds.

(2) Since $s \in K^\circ$, d_K is embedded on the external face of $\Psi_{H'}$. Observe that $\bar{H}$ can be obtained from H' substituting the edge (u, v) with a path homeomorphic to a valley. Thus, by Lemma 8 and by Fact 8, the lemma holds.

(3) Observe that $\bar{H}$ can be obtained from H' by reversing edge (u, v) and by direct subdivision. Thus, following the proof of case 4.3 of Lemma 9, and by Fact 8, the lemma holds.

Suppose K, H, K', and H' satisfy the conditions of Lemma 11. Let P_K and P_H be the paths associated with d_K and d_H, respectively (see Figures 13(a), (b)). Except for the common endpoints u and v, P_K and P_H are disjoint paths of G, since they lie in different components. Thus, $C = P_K \cup P_H$ is a simple (undirected) cycle of G.

Let $\bar{K} = K \cup P_H$ and $\bar{H} = H \cup P_K$ (see Figures 13(c), (d)). Since K' and H' are upward planar, by the previous lemma, $\bar{K}$ and $\bar{H}$ are upward planar. Let $\Psi_{\bar{K}}$ and $\Psi_{\bar{H}}$ be two upward embeddings of $\bar{K}$ and $\bar{H}$, respectively, and let $\alpha_{\bar{K}}$ and $\alpha_{\bar{H}}$ be the corresponding external faces. Now, let $K^*(H^*)$ be the subgraph of $\bar{K}(\bar{H})$ embedded inside C in $\Psi_{\bar{K}}(\Psi_{\bar{H}})$ (see Figures 13(e), (f)). We have the following lemma.

LEMMA 13. *Let s^* be a source of K^* (H^*). Then $s^* \in C$.*

Proof. Suppose $s^* \notin C$. Then s^* is a source of $\bar{K}$ embedded inside C in $\Psi_{\bar{K}}$. Since $\bar{K}$ is an sT-digraph, s^* is the only source of $\bar{K}$ and is embedded on the external face of every upward embedding of $\bar{K}$, and thus cannot be embedded inside C in $\Psi_{\bar{K}}$, which is a contradiction. □

LEMMA 14. *The digraph K^* (H^*) is an expanded sT-digraph.*

Proof. Since K^* is a subgraph of G, then it is acyclic and expanded. By Lemma 13, all of the sources of K^* belong to $C = P_K \cup P_H$. In order to prove the existence of a single source, we have to consider the "shapes" of P_K and P_H. Observe that P_K (P_H) can be homeomorphic to an edge, a peak, a valley, or a zig-zag. If both P_K and P_H are homeomorphic to the edge (u, v) $((v, u))$, then $s(K^*)$ is either u or v. Suppose now that P_K is not homeomorphic to an edge. We consider the following cases.

1. P_K is homeomorphic to a peak. Clearly, if P_H is homeomorphic to (u, v) or (v, u) or a valley, then C has only one source. We now show that P_H is not homeomorphic to a peak or to a zig-zag. Since P_K is a peak, then K is in case 1 or in case 3b.2 of the proof of Lemma 9.

Suppose P_H is a peak. Then H is in case 1 or in case 3b.2.

(a) P_K as in case 1. If H is in case 1, then G contains two sources, which is a contradiction. If H is in case 3b.2, then v is not a source of K, which is a contradiction.

(b) P_K as in case 3b.2. If H is in case 1, then the proof is as above. If H is in case 3b.2, then v is internal both in H and in K, which is a contradiction.

Suppose P_H is homeomorphic to a zig-zag. Then H is in case 2.b or in case 3b.1.

(a) P_K as in case 1. If H is in case 2.b or in case 3b.1, then $s(G) \in H^\circ$. Since u is source both in K and in H then G contains two sources, which is a contradiction.

(b) P_K as in case 3b.2. The proof is as above.

2. P_K homeomorphic to a zig-zag (w.l.o.g., we can suppose $a \equiv u$ and $b \equiv v$). Clearly, if P_H is homeomorphic to (v, u), then C has only one source. If P_H is

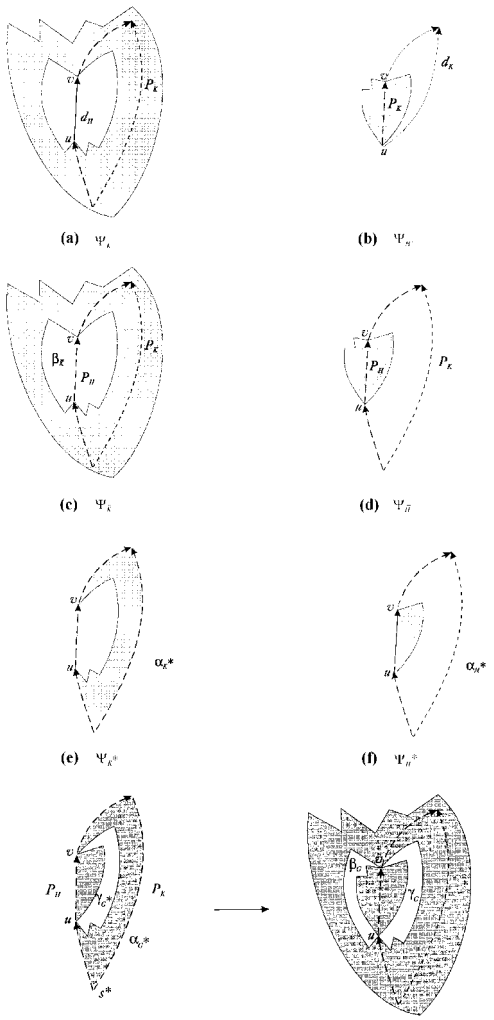


FIG. 13. *Construction of* Ψ_G.

homeomorphic to a peak, the proof is as for the case P_H homeomorphic to a zig-zag and P_K homeomorphic to a peak. We now show that P_H is not homeomorphic to the edge (u, v), to a valley, or to a zig-zag. Since P_K is a zig-zag, then K is in case 2(b) or case 3b.1; thus $s(G) \in K^\circ$ and u is a source of K.

Suppose P_H is homeomorphic to the edge (u, v). Then H is in case 2(a) or case 3(a) or case 4.1. If H is in case 2(a) or case 3(a), then u is a source of G and G contains two sources, which is a contradiction. If H is in case 4.1, then $s(G) \in H^\circ$, which is a contradiction.

Suppose P_H is homeomorphic to a valley. Then H is in case 4.2 and $s(G) \in H^\circ$, which is a contradiction.

Suppose P_H is homeomorphic to a zig-zag. Then H is in cases 2(b) and 3b.1 and $s(G) \in H^\circ$, which is a contradiction.

3. P_K is homeomorphic to a valley. Clearly, if P_H is homeomorphic to the edge (u, v), to the edge (v, u), or to the peak, then C has only one source. If P_H is

homeomorphic to a zig-zag, the proof is as for the case where P_H is homeomorphic to a valley and P_K is homeomorphic to a zig-zag. We now show that P_H is not homeomorphic to a valley.

Since both P_K and P_H are homeomorphic to a valley, then both K and H are in case 4.2; thus $s(G) \in K^\circ$ and $s(G) \in H^\circ$, which is a contradiction. □

The proof of the next lemma can be found in [3, 4].

LEMMA 15. *Let G be digraph, let Ψ_G be a candidate planar embedding of G, and let face $\delta \in \Psi_G$. Finally, let $\mathcal{A}$ be an assignment of the sinks and the sources of G to the faces of Ψ_G. If $|A(f)| = c(f)$ for each face $f \in \Psi_G$, with $f \neq \delta$, then $|A(\delta| = c(\delta)$.*

Suppose G, K, H, K', and H' satisfy the conditions of Lemma 11. In the following we give a constructive procedure to derive an embedding Ψ_G of G from two upward embeddings $\Psi_{H'}$ and $\Psi_{K'}$ of H' and K', respectively. We then show Ψ_G to be upward.

Let $\bar{K}$, $\bar{H}$, K^*, H^*, Ψ_{K^*}, Ψ_{H^*} be as defined for Lemmas 13 and 14 and let $\Psi_{K^*} \subseteq \Psi_{\bar{K}}(\Psi_{H^*} \subseteq \Psi_{\bar{H}})$ be the upward embedding of K^* (H^*) contained in $\Psi_{\bar{K}}$ ($\Psi_{\bar{H}}$). Denote by α_{K^*} and α_{H^*} the external faces of Ψ_{K^*} and Ψ_{H^*}, respectively (see Figures 13(e), (f)). Recall that $\alpha_{K^*} = \alpha_{H^*} = C$.

Let $G^* = K^* \cup H^*$ (see Figure 13(g)). Ψ_{G^*} is a planar embedding of G^* such that $\Psi_{K^*} \subset \Psi_{G^*}$, $\Psi_{H^*} \subset \Psi_{G^*}$, and P_K and P_H lie on the external face α_{G^*} of Ψ_{G^*} (i.e., $\alpha_{G^*} = C$).

We denote by γ_{G^*} the other face of Ψ_{G^*} (besides α_{G^*}) containing both edges of K^* and edges of H^*.

LEMMA 16. *Let G^* and Ψ_{G^*} be defined as above. Then*

(i) *G^* is an expanded sT-digraph.*

(ii) *Ψ_{G^*} is an upward embedding.*

Proof. (i) Since G^* is a subgraph of G, then G^* is expanded and acyclic. Since G^* is acyclic, it contains at least one source. Suppose G^* contains two sources s_1 and s_2. Since K^* (H^*) is an sT-digraph, both s_1 and s_2 cannot belong to K^* (H^*). Thus, w.l.o.g., $s_1 \in K^*$ and $s_2 \in H^*$. Furthermore, by Lemma 13, s_1 and s_2 lie on cycle C, so they are both contained in K^* and H^*, which is a contradiction. In the following, we denote by s^* the source of G^*.

(ii) We derive an upward consistent assignment $\mathcal{A}_{G^*}$ from the upward consistent assignments $\mathcal{A}_{K^*}$ and $\mathcal{A}_{H^*}$ associated with Ψ_{K^*} and Ψ_{H^*}.

First note that $A_{K^*}(\alpha_{K^*}) = A_{H^*}(\alpha_{H^*})$. In fact, let T^* be the set of sink-switches of $C = \alpha_{K^*} = \alpha_{H^*}$. Since Ψ_{K^*} and Ψ_{H^*} are upward embeddings of sT-digraphs, by Fact 2, each sink-switch on α_{K^*} and α_{H^*} is a sink of K^* and H^*, respectively, and they are all assigned to α_{H^*} and α_{K^*}. We have that $A_{K^*}(\alpha_{K^*}) = A_{H^*}(\alpha_{H^*}) = T^* \cup \{s^*\}$.

For each face $f \in \Psi_{G^*}$, with $f \neq \gamma_{G^*}$ and $f \neq \alpha_{G^*}$, we have that f belongs to Ψ_{K^*} or f belongs to Ψ_{H^*}, but not both. It is trivial to see that the following assignment to the faces of $\Psi_{G^*} - \{\gamma_{G^*}\}$ is feasible (i.e., the number of vertices assigned to each face equals the capacity of the face):

- $A_{G^*}(\alpha_{G^*}) = A_{K^*}(\alpha_{K^*}) = A_{H^*}(\alpha_{H^*})$;
- $A_{G^*}(f) = A_{K^*}(f)$, for $f \neq \alpha_{G^*}$ and $f \in \Psi_{K^*}$;
- $A_{G^*}(f) = A_{H^*}(f)$, for $f \neq \alpha_{G^*}$ and $f \in \Psi_{H^*}$.

Observe that all the sinks of Ψ_{G^*} not assigned by the above assignment lie on face γ_{G^*}, and so they can be assigned to it in $\mathcal{A}^*_G$. Since Ψ_{G^*} is a candidate embedding and, for each face $f \in \Psi_{G^*} - \{\gamma_{G^*}\}$, it is $|A_{G^*}(f)| = c(f)$, by Lemma 15 we have that $A_{G^*}(\gamma_{G^*}) = c(\gamma_{G^*})$. Thus, A_{G^*} is an upward consistent assignment and G^* is

upward planar. □

We are now able to give the proof of Lemma 11.

Proof of Lemma 11. A planar embedding Ψ_G of G can be obtained from the upward embeddings $\Psi_{\bar{K}}$ and $\Psi_{\bar{H}}$ in such a way that $\Psi_{G^*} \subseteq \Psi_G$. Each face of Ψ_G belongs either to $\Psi_{\bar{K}}$ or to $\Psi_{\bar{H}}$, except for two faces which share edges both of $\bar{K}$ and $\bar{H}$. One of these two faces coincides with face γ_{G^*} of Ψ_{G^*}, and thus it is internal in Ψ_{G^*}; we denote it by γ_G and denote the other face by β_G (see Figure 13(h)). Conversely, all the faces of $\Psi_{\bar{K}}$ ($\Psi_{\bar{H}}$), except for the two faces sharing P_H (P_K), belong to Ψ_G.

We now derive an assignment $\mathcal{A}_G$ from the upward consistent assignments $\mathcal{A}_{\bar{K}}$, $\mathcal{A}_{\bar{H}}$, and $\mathcal{A}_{G^*}$ in the following way:

- $A_G(\gamma_G) = A_{G^*}(\gamma_{G^*})$;
- $A_G(f) = A_{\bar{K}}(f)$, for $f \in \Psi_{\bar{K}}$;
- $A_G(f) = A_{\bar{H}}(f)$, for $f \in \Psi_{\bar{H}}$;
- it is easy to see that all remaining sinks and (eventually) the source of G stay on face β_G and so they are assigned to β_G in $\mathcal{A}_G$.

In order to prove that $\mathcal{A}_G$ is upward consistent we have to show that

(i) every sink and the source of G is assigned to exactly one face of Ψ_G;
(ii) every internal vertex of G is not assigned to any face of Ψ_G;
(iii) the number of vertices assigned to each face equals the capacity of the face.

Observe first that since P_K is embedded in the external face $\alpha_{\bar{H}}$ of $\Psi_{\bar{H}}$, $\alpha_{\bar{H}}$ is not a face of Ψ_G. Moreover, since γ_{G^*} is not the external face of Ψ_{G^*}, u and v are not assigned to it in $\mathcal{A}_{G^*}$ (they both lie on the external face) and, in turn, in $\mathcal{A}_G$.

We first prove (i). It is easy to see that each sink (source) of G is assigned to at least one face. We have to prove that each sink (source) of G is assigned to at most one face. Let x be a vertex assigned to two faces f_1 and f_2 in $\mathcal{A}_G$ ($f_1 \neq f_2$). From the definition of $\mathcal{A}_G$, $f_1 \in \Psi_{\bar{K}}$ and $f_2 \in \Psi_{\bar{H}}$. This is not possible if $x \neq u$ or $x \neq v$. If $x = u$ ($x = v$) then it is assigned to the external face $\alpha_{\bar{H}} = f_2$ of $\Psi_{\bar{H}}$ in $\mathcal{A}_{\bar{H}}$ and thus $f_2 \notin \Psi_G$, which is a contradiction.

(ii) Let x be an internal vertex of G and suppose it is assigned to a face f of Ψ_G. Again $x = u$ or $x = v$. Assume, w.l.o.g., $x = v$. Observe that f is not a face of $\Psi_{\bar{H}}$ (v is eventually assigned to its external face, which is not a face of Ψ_G). Furthermore, $f \neq \gamma_G$, since neither u nor v is assigned to it in $\mathcal{A}_G$. So f is a face of $\Psi_{\bar{K}}$.

Let us denote by $\beta_{\bar{K}}$ the face of $\Psi_{\bar{K}}$ sharing the path P_K and not embedded inside cycle C (see Figure 13(c)). Face $\beta_{\bar{K}}$ is not a face of Ψ_G, and thus $f \neq \beta_{\bar{K}}$. Since v is not assigned to $\beta_{\bar{K}}$ in $\mathcal{A}_{\bar{K}}$, $\beta_{\bar{K}}$ is internal in $\Psi_{\bar{K}}$. So, both $\beta_{\bar{K}}$ and $\gamma_{\bar{K}}$ are internal faces of $\Psi_{\bar{K}}$ and hence Ψ_H is embedded in a face of Ψ_K in Ψ_G. This implies that $s(\bar{K}) = s(G) \in \bar{K}$.

Suppose now that $x = v$ is a sink in $\bar{K}$ and v is not a sink in G.

Since v is a sink of $\bar{K}$, then P_H has an incoming edge into v, and P_H is homeomorphic to the edge (u, v) or to a zig-zag or to a valley. If P_H is homeomorphic to edge (u, v), then H is in case 2(a), 3(a), or 4.1 of the proof of Lemma 9. In case 2(a), v is a sink of H and thus u is a sink of G, which is a contradiction. In case 3(a), v is a source of K and thus it is not a sink in $\bar{K}$, which is a contradiction. If H is in case 4.1 then $s \in H^\circ$, which is a contradiction.

If P_H is homeomorphic to a valley, then H is in case 4.2 and $s(G) \in H^\circ$, which is a contradiction.

If P_H is homeomorphic to a zig-zag, then H is in case 2(b) or 3b.2. In both cases, $s(G) \in H^\circ$, which is a contradiction.

(iii) Since G is an expanded digraph, every planar embedding is candidate. By Lemma 15, the capacity equation for face β_G is satisfied, and $\mathcal{A}_G$ is upward consistent. □

Lemma 11 refers to digraph G. We extend the result to a minor $\tilde{G}$ of G. This is done by showing that, under certain restrictions, the directed-virtual-edges substituting components can be chosen independently one from another. In particular, this is true if the source s of G belongs to its minor. Note that in this case a minor of a minor of G is a minor of G.

LEMMA 17. *Let $\tilde{G}$ be a minor of G such that $s(\tilde{G}) = s(G)$. Let $\{u, v\}$ be a split pair of $\tilde{G}$ such that $\{u, v\}$ is also a split pair of G. Let $\tilde{K}$ be a component of $\tilde{G}$ w.r.t. $\{u, v\}$ and let K be the corresponding component of G, i.e., K is obtained from $\tilde{K}$ by replacing each directed-virtual-edge of $\tilde{K}$ with its associated component of G. Then it is $d(\tilde{K}, \tilde{G} - \tilde{K}) = d(K, G - K)$.*

Proof. In the following we denote by H the digraph $G - K$ and by $\tilde{H}$ the digraph $\tilde{G} - \tilde{K}$. Observe that, for each component J of G such that $J \notin \tilde{G}$ (i.e., J is substituted by its directed-virtual-edge), since $s(G) \in \tilde{G}$, then $s(G) \notin J^\circ$.

We examine the following four cases corresponding to the substitution Rules 1–4 applied to components K and $\tilde{K}$.

1. Since u and v are sources of K, by Fact 9, u and v are sources of $\tilde{K}$. Then $d(\tilde{K}, \tilde{H})$ is a peak and the lemma holds.

2(a) u is a source of K, v is a sink of K, and $s(G) \notin K^\circ$. By Fact 9, u is a source of $\tilde{K}$. Since $s(\tilde{G}) = s(G)$ then $s(G) \notin K^\circ$. We now show that v is a sink of $\tilde{K}$. In fact, v is a sink of all the components having v as a pole. By the substitution rules, whenever a component $J \in K$ has v as a sink then the associated directed-virtual-edge has an edge incoming into v except for Rules 2(b) and 4(b). But, in both cases, $s(G) \in J^\circ$, contradicting that $s \notin K_\circ$. Then Rule 2(a) is applied to $\tilde{K}$ and $d(\tilde{K}, \tilde{H})$ is the edge (u, v).

2(b) u is a source of K, v is a sink of K, and $s(G) \in K^\circ$. By Fact 9, u is a source of $\tilde{K}$. Furthermore, $s(\tilde{G}) \in \tilde{K}^\circ$.

If v is a sink of $\tilde{K}$, then Rule 2(b) is applied to $\tilde{K}$ and $d(\tilde{K}, \tilde{H})$ is a peak.

If v is internal of $\tilde{K}$, then Rule 3(b) is applied to $\tilde{K}$ and $d(\tilde{K}, \tilde{H})$ is a peak.

If v is a source of $\tilde{K}$, then $\tilde{K}$ has three sources and the minor $\tilde{G}$ is not an sT-digraph (it contains at least two sources), which is a contradiction.

3(a) u is a source of K, v is internal of K, $s(G) \in H$, and v is a source of H. By Fact 9, u is a source of $\tilde{K}$ and v is a source of $\tilde{H}$.

If v is internal in $\tilde{K}$, then Rule 3(a) is applied to $\tilde{K}$ and $d(\tilde{K}, \tilde{H})$ is the edge (u, v).

If v is a sink of $\tilde{K}$, then Rule 2(a) is applied to $\tilde{K}$ and $d(\tilde{K}, \tilde{H})$ is the edge (u, v).

If v is a source of $\tilde{K}$, then v is a source of $\tilde{G}$. Since $s(\tilde{G}) = s(G) \neq v$, then $\tilde{G}$ has two sources, which is a contradiction.

3(b) u is a source of K and v is internal in K, $s(G) \in K^\circ$ or v is not a source of H.

By Fact 9, u is a source of $\tilde{K}$. Two cases are possible.

(i) $s(G) \in K^\circ$. Then v is the only source of H. In fact, if u is a source of H, then u is a source of G, which is a contradiction. By Fact 9, v is a source of $\tilde{H}$.

If v is internal in $\tilde{K}$, then Rule 3(b) is applied to $\tilde{K}$ and $d(\tilde{K}, \tilde{H})$ is a peak.

If v is a sink of $\tilde{K}$, then Rule 2(b) is applied to $\tilde{K}$ and $d(\tilde{K}, \tilde{H})$ is a peak.

If v is a source of $\tilde{K}$, then v is a source of $\tilde{G}$. Since $s(\tilde{G}) = s(G) \neq v$, then $\tilde{G}$ has two sources, which is a contradiction.

(ii) v is not a source of H and $s(G) \in H$. Since G is expanded, v is a sink of H.

If v is a source of $\tilde{K}$, then Rule 1 is applied to $\tilde{K}$ and $d(\tilde{K}, \tilde{H})$ is a peak.

If v is internal in $\tilde{K}$ and v is not a source of $\tilde{H}$, then Rule 3(b) is applied to $\tilde{K}$ and $d(\tilde{K}, \tilde{H})$ is a peak. If v is a source of $\tilde{H}$, there exists a component J of H, with $J \notin \tilde{H}$, having v and u_J as poles, whose directed-virtual-edge is either a peak or the edge (v, u_J). Observe that v is a sink of J and u_J is the source of J (since $s(G) \notin J^\circ$). But in this case, Rule 2(a) is applied to J and $d(J, G - J)$ is the edge (u_J, v), which is a contradiction.

If v is a sink of $\tilde{K}$, since v is internal in K, there exists a component J, with $J \in K$ and $J \notin \tilde{K}$, having v and u_J as poles, whose directed-virtual-edge is the edge (u_J, v). Furthermore, v is an internal vertex or a source of J. If v is a source of J then $d(J, G - J)$ is not the edge (u_J, v). If v is internal in J, then Rules 3(a) or 4(b) must be applied in order to have $d(J, G - J) = (u_J, v)$. Rule 3(a) implies that v is a source of $G - J$; hence v is a source of K, which is a contradiction. Rule 4(a) implies that $s(G) \in J^\circ$, which is a contradiction.

4. u and v are not sources of K. $s \in K^\circ$. Either u or v (or both) is a source of H. Suppose, w.l.o.g., that u is a source of H. u is not a source of $\tilde{K}$; otherwise the minor $\tilde{G}$ has two sources. By Fact 9, u is a source of $\tilde{H}$.

If v is not a source of $\tilde{K}$, Rule 4(a) can be applied and $d(\tilde{K}, \tilde{H})$ is the edge (u, v).

If v is a source of $\tilde{K}$, there exists a component J of K, with $J \notin \tilde{K}$, having v and u_J as poles, such that v is a nonsource of J, and its directed-virtual-edge is either a peak or the edge (v, u_J). Observe that, since $s(G) \notin J^\circ$, u_J is the source of J. If v is a sink of J, Rule 2(a) is applied to J and $d(J, G - J)$ is the edge (u_J, v), which is a contradiction. If v is internal in J, Rule 3(b) is applied and, since $s(G) \notin J^\circ$, v is a sink of $G - J$ and hence it is a sink in all components of $K - J$. Now, v is a source in the digraph $\tilde{K} - d(J, G - J)$. Then there exists a component Z of K, with $Z \neq J$ and $Z \notin \tilde{K}$, having v and u_Z as poles, such that v is a source in $d(Z, G - Z)$. Since v is a sink of Z and $s(G) \notin Z^\circ$ (since $s(G) \in \tilde{K}$), u_Z is a source of Z and then Rule 2(a) is applied. But then $d(Z, G - Z)$ is the edge (u_Z, v), which is a contradiction. □

We are now able to prove the sufficiency part of Theorem 3.

Proof of sufficiency of Theorem 3. Let $\mathcal{T}$ be the rooted *SPQR*-tree associated with the graph G, and let $\mu_1, \ldots, \mu_m$ be the sequence of nodes of $\mathcal{T}$ deriving from a depth-first-search (DFS) visit of $\mathcal{T}$, starting at its root. Let $Skel(\mu_i), i = 1, \ldots, m$ be the sT-skeleton associated with $\mu, \ldots, \mu_m$.

For each node μ_i, let d_i^c be the directed virtual edge of μ_i in the skeleton associated with the parent of μ_i, and let d_i^p be the directed-virtual-edge of the parent of μ_i in $Skel(\mu_i)$. Clearly, if $\mu_i = \mu_1$ is the root, then $d_i^p = d_1^p = \{\emptyset\}$. Finally, let $\tilde{G}_i = Skel(\mu_1) - d_2^c \cup (Skel(\mu_k) - d_2^p) - d_3^c \cup (Skel(\mu_3) - d_3^p) - \cdots - d_i^c \cup (Skel(\mu_i) - d_i^p)$.

We show that $\tilde{G}_i$ is a minor of G, with $s(G) \in \tilde{G}_i$, and that $\tilde{G}_i$ is upward planar.

For $i = 1$ we have that $\tilde{G}_1 = Skel(\mu_1)$ and the claim trivially holds. Suppose that $\tilde{G}_{l-1}$ is a minor of G with $s(G) \in \tilde{G}_{l-1}$, and that $\tilde{G}_{l-1}$ is upward planar. We show that $\tilde{G}_l$ is a minor of G, with $s(G) \in \tilde{G}_l$, and that $\tilde{G}_l$ is upward planar.

Let H be the pertinent graph of d_l^c and K be the pertinent graph of d_l^p. Recall that $K \cup H = G$ and that K and H share exactly two vertices. Moreover $d_l^p = d(K, H)$ and $d_l^c = d(H, K)$.

Let $J_1, \ldots, J_q$ be components of G contained in K, such that

$$\begin{aligned}\tilde{G}_{l-1} &= G - H - J_1 - \cdots - J_q \cup d(H, K) \cup d(J_1, G - J_1) \cup \cdots \cup d(J_q, G - J_q) \\ &= K - J_1 - \cdots - J_q \cup d(H, K) \cup d(J_1, G - J_1) \cup \cdots \cup d(J_q, G - J_q),\end{aligned}$$

and let $Z_1, \ldots, Z_r$ be split components of G contained in H, such that

$$\begin{aligned} \mathit{Skel}(\mu_l) &= G - K - Z_1 - \cdots - Z_r \cup d(K,H) \cup d(Z_1, G - Z_1) \cup \cdots \cup d(Z_r, G - Z_r) \\ &= H - Z_1 - \cdots - Z_q \cup d(K,H) \cup d(Z_1, G - Z_1) \cup \cdots \cup d(Z_r, G - Z_r). \end{aligned}$$

The digraph $\tilde{G}_l = K \cup H - J_1 - \cdots - J_q - Z_1 - \cdots - Z_q \cup d(J_1, G - J_1) \cup \cdots \cup d(J_q, G - J_q) \cup d(Z_1, G - Z_1) \cup \cdots \cup d(Z_r, G - Z_r)$ is a minor of G. In fact, since $J_i \in K$, $i = 1, \ldots, q$, and $Z_t \in H$, $t = 1, \ldots, r$, it follows that J_i and Z_t do not share any edge, for $i = 1, \ldots, q$ and $t = 1, \ldots, r$.

Since $s(G) \in \tilde{G}_{l-1}$, then $s(G) \in \tilde{G}_l$.

Let $\tilde{K} = \tilde{G}_{l-1} - d_l^c$ and $\tilde{H} = \mathit{Skel}(\mu_l) - d_l^p$. Clearly $\tilde{G}_l = \tilde{K} \cup \tilde{H}$. By Lemma 17, $d(\tilde{K}, \tilde{H}) = d(K,H) = d_l^p$ and $d(\tilde{H}, \tilde{K}) = d(H,K) = d_l^c$. Since $\tilde{K} \cup d(\tilde{H}, \tilde{K}) = \tilde{G}_{l-1}$ is upward planar and $\tilde{H} \cup d(\tilde{K}, \tilde{H}) = \mathit{Skel}(\mu_l)$ is upward planar with $d(\tilde{K}, \tilde{H}) = d_l^p$ embedded on the external face, then by Lemma 11, $\tilde{G}_l$ is upward planar.

We now show that $\tilde{G}_m = G$. By induction $\tilde{G}_m$ is a minor of G. Suppose $\tilde{G}_m \neq G$; then there exists a component J of G such that $J \notin \tilde{G}_m$ and $d(J, G - J) \in \tilde{G}_m$. Let μ_j be the node of $\mathcal{T}$ such that $d(J, G - J) \in \mathit{Skel}(\mu_j)$. $d(J, G - J)$ is associated with either the parent of μ_j or one of the children of μ_j. Since the tree $\mathcal{T}$ has been entirely visited then $d(J, G - J)$ has been substituted, and thus $d(J, G - J) \notin \tilde{G}_m$, which is a contradiction. □

7. Algorithm for general single-source digraphs. Let G be a biconnected single-source digraph. In this section we present an algorithm for testing whether G is upward planar.

ALGORITHM. *Test.*

1. Construct the expansion G' of G.

2. Test whether G' is planar. If G' is not planar, then return "not-upward-planar" and stop; else, construct an embedding for G'.

3. Test whether G' is acyclic. If G' is not acyclic, then return "not-upward-planar" and stop.

4. Construct the SPQR-tree $\mathcal{T}$ of G' and the skeletons of its nodes.

5. For each virtual edge e of a skeleton, classify each endpoint of e as a source, sink, or internal vertex in the pertinent digraph of e. Also, determine if the pertinent digraph of e contains the source.

6. For each node μ of $\mathcal{T}$, compute the sT-skeleton of μ.

7. For each R-node μ of $\mathcal{T}$

(a) test whether the sT-skeleton of μ is upward planar by means of algorithm *Embedded-Test*. If *Embedded-Test* returns "not-upward-planar," then return "not-upward-planar" and stop.

(b) mark the virtual edges of the skeleton of μ whose endpoints are on the external face in some upward drawing of the sT-skeleton of μ.

(c) for each unmarked virtual edge e of the skeleton of μ, constrain the tree edge of $\mathcal{T}$ associated with e to be directed towards μ.

(d) if the source is not in $\mathit{skeleton}(\mu)$, let ν be the node neighbor of μ whose pertinent digraph contains the source, and constrain the tree edge (μ, ν) to be directed towards ν.

8. Determine whether $\mathcal{T}$ can be rooted at a Q-node in such a way that orienting edges from children to parents satisfies the constraints of steps 7(c)–(d). If such a rooting exists then return "upward-planar"; else return "not-upward-planar."

For single-source digraphs that are not biconnected we apply the above algorithm to each biconnected component.

THEOREM 4. *Upward planarity testing of a single-source digraph with n vertices can be done in $O(n)$ time using $O(n)$ space.*

Proof. Steps 1 and 3 can be trivially performed in $O(n)$ time. Planarity testing in step 2 can also be done in $O(n)$ time [18]. The construction of the SPQR-tree and the skeletons of its nodes (step 4) takes time $O(n)$ using a variation of the algorithm of [17]. The preprocessing of step 5 consists essentially of a visit of $\mathcal{T}$ and can be done in $O(n)$ time. Let n_μ be the number of vertices of the skeleton of μ. The information collected in step 5 allows us to perform step 6 in $O(n)$ time and step 7(d) in $O(n_\mu)$ time. By Theorem 2, step 7(a) takes $O(n_\mu)$ time. The output of step 7(a) allows us to perform steps 7(b)–(c) in $O(n_\mu)$ time. Since $\sum_\mu n_\mu = O(n)$, the total complexity of step 7 is $O(n)$. Finally, step 8 consists of a visit of $\mathcal{T}$ and takes $O(n)$ time. □

To parallelize Algorithm *Test*, we need an efficient way of testing in parallel whether a planar single-source digraph with n vertices is acyclic. For this purpose, we can use the algorithm of [20], which runs in $O(\log^3 n)$ time on a CRCW PRAM with n processors. However, the particular structure of planar single-source digraphs allows us to perform this test optimally. The following characterization is inspired by some ideas in [20].

Let G be an embedded, expanded, planar single-source digraph. The *clockwise subgraph* of G is obtained by taking the first incoming edge of each internal vertex, in clockwise order. The *counterclockwise subgraph* of G is similarly obtained by taking the first incoming edge of each internal vertex, in counterclockwise order. Such subgraphs of G have all vertices with in-degree 1 or 0.

THEOREM 5. *An embedded, expanded single-source digraph G is acyclic if and only if both the clockwise and counterclockwise subgraphs of G are acyclic.*

Proof. The only-if part is trivial. For the if part, assume for contradiction that G is not acyclic, and consider an arbitrary drawing of G with the prescribed embedding and with the source on the external face. We will show the existence of a cycle in either the clockwise or counterclockwise subgraph. Let γ be a cycle of G that does not enclose any other cycle. Since the source of G must be outside γ, all the edges incident on vertices of γ and inside γ must be outgoing edges. Hence, γ is contained in the clockwise or counterclockwise subgraph depending on whether it is a clockwise or counterclockwise cycle. □

The structure of each connected component of the clockwise and counterclockwise subgraphs is either a source tree, or a collection of source trees with their roots connected in a directed cycle. Hence, one can test whether such subgraphs are acyclic using standard parallel techniques. Since expansion preserves acyclicity, we have the following theorem.

THEOREM 6. *Given an embedded planar single-source digraph G with n vertices, one can test if G is acyclic in $O(\log n)$ time with $n/\log n$ processors on an EREW PRAM.*

By applying the result of Theorem 6 and various parallel techniques (in particular [14, 28, 27]) we can efficiently parallelize algorithm *Test*.

THEOREM 7. *Upward planarity testing of a single-source digraph with n vertices can be done in $O(\log n)$ time on a CRCW PRAM with $n \log\log n/\log n$ processors using $O(n)$ space.*

As a consequence of Theorems 1 and 3, algorithm *Test* can be easily extended such that if the n-vertex digraph G is found to be upward planar, a planar st-digraph

G' with $O(n)$ vertices is constructed that contains G as a subdigraph. Hence, by applying the planar polyline upward drawing algorithm of Di Battista, Tamassia, and Tollis [12] to G' and then removing the vertices and edges of G' that are not in G, we obtain a planar polyline upward drawing of G.

THEOREM 8. *Algorithm* Test *can be extended so that it constructs a planar polyline upward drawing if the digraph is upward planar. The complexity bounds stay unchanged.*

7.1. Examples of application of algorithm *Test*. In this subsection we use two examples to illustrate the behavior of algorithm *Test* in performing the upward planarity testing. In the first example, the algorithm is applied to a graph which is not upward drawable; in the second example, the algorithm is applied to an upward drawable graph.

Example 1. In this example we consider the graph G of Figure 2a. We apply algorithm *Test* step by step.

1. The expansion graph G' of G is shown in Figure 14.

2. By a simple inspection it is possible to verify that G' is planar.

3. Again, by a simple inspection, it is possible to verify that G' is acyclic.

4. The SPQR-tree $\mathcal{T}$ of G' is shown in Figure 15. The skeletons of the nodes of $\mathcal{T}$ are shown in Figure 16 (the skeletons of the Q nodes are omitted).

5. Consider, for example, the virtual edge (2, 15) in skeleton μ_1 of Figure 16, which is the virtual edge of μ_4. The pertinent graph of μ_4 is subgraph G'_4 of G' induced by the node set $V(G') - \{13, 14, 24, 25\}$. By simple inspection, it is possible to verify that node 2 is internal in G'_4, while node 15 is a sink in G'_4. In addition, the source 1 of G' is contained in G'_4. In the same way, all other endnodes of virtual edges can be classified.

6. The sT-skeletons corresponding to the skeletons of Figure 16 are shown in Figure 17. Consider, for example, the sT-skeleton of μ_2. The pertinent graph G'_1 associated with the virtual edge (2, 13) of the skeleton of μ_2 (Figure 16) is the subgraph of G' induced by the node set $V(G') - \{24\}$. Using the classification performed in the preceding step, it is easy to verify that: (i) 2 is internal in G'_1 and 13 is a source in G'_1. In addition, the source 1 of G' belongs to G'°_1, hence, by Rule 3b the directed-virtual-edge associated with G'_1 is a peak. The pertinent graph associated with the virtual edge (2, 24) of μ_2 is the directed edge (2, 24). If is easy to see that the directed-virtual-edge associated with an edge u, v, is the edge (u, v) (Rule 3a). Thus, the directed-virtual-edge associated with the directed edge 2, 24 is again (2, 24). The same holds for edge (24, 13).

7. Now we perform Steps 7(a)–7(d) on all the R-nodes of $\mathcal{T}$.

(a) In Figure 18 we show the face-sink graphs associated with the sT-skeletons of the R-nodes μ_1, μ_5, and μ_{12} (the vertices associated with the faces are represented by squares). It is easy to verify that they all satisfy the conditions of Theorem 1; thus the algorithm *Embedded Test* will return "upward-planar" for every R-node. It also returns, for each R-node, the set of faces that can be external faces in an upward drawing of the associated sT-skeleton. In the figure, the nodes associated with these faces are indicated by black squares.

(b) By inspection of Figure 18, the unmarked edges are the following:

- μ_1: $\{(13, 14)\}$.
- μ_5: $\{(2, 5), (5, 8), (5, 11), (6, 8), (7, 8), (9, 11), (10, 11)\}$.
- μ_{12}: $\{(1, 3), (2, 3), (3, 4)\}$.

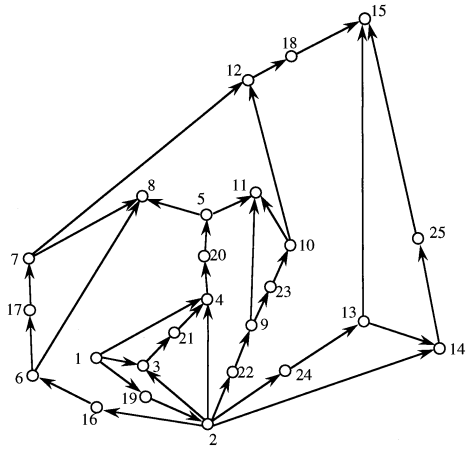


FIG. 14. *Expansion graph of graph G of Figure* 2a.

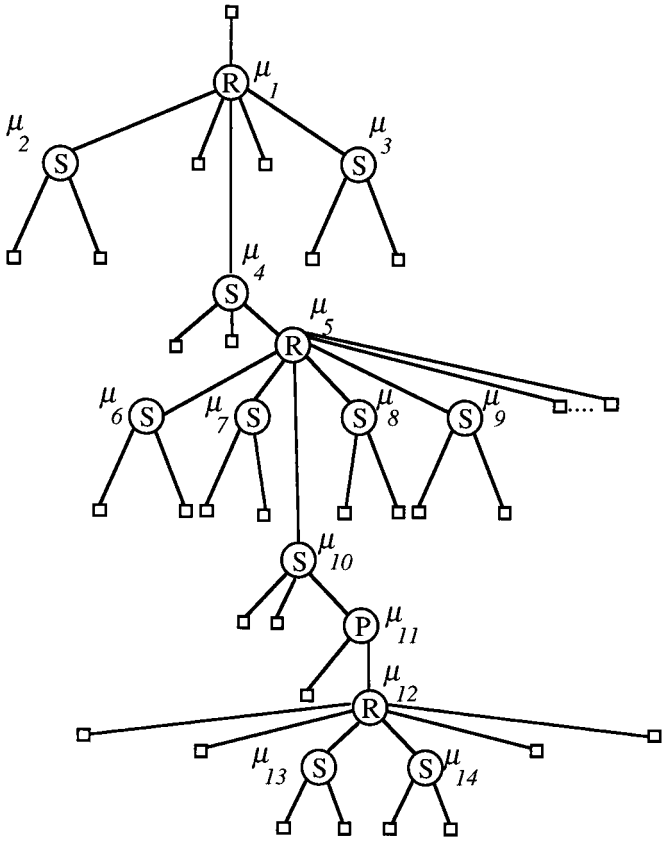


FIG. 15. *The SPQR-tree* $\mathcal{T}$ *of* G'.

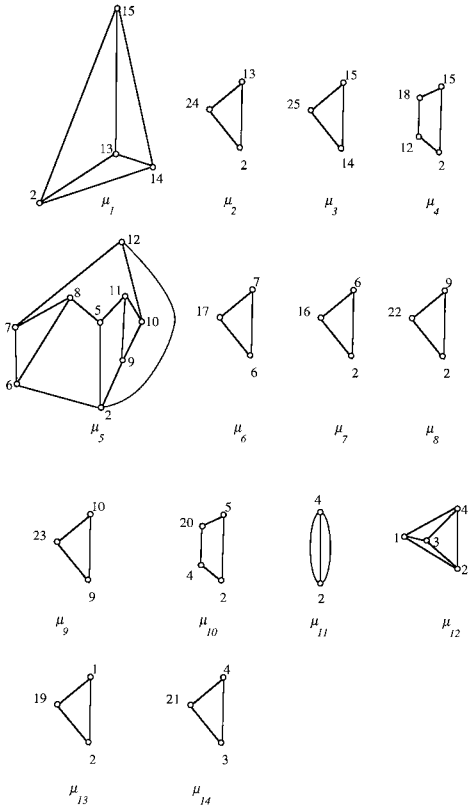


FIG. 16. *The skeletons of the nodes of* $\mathcal{T}$.

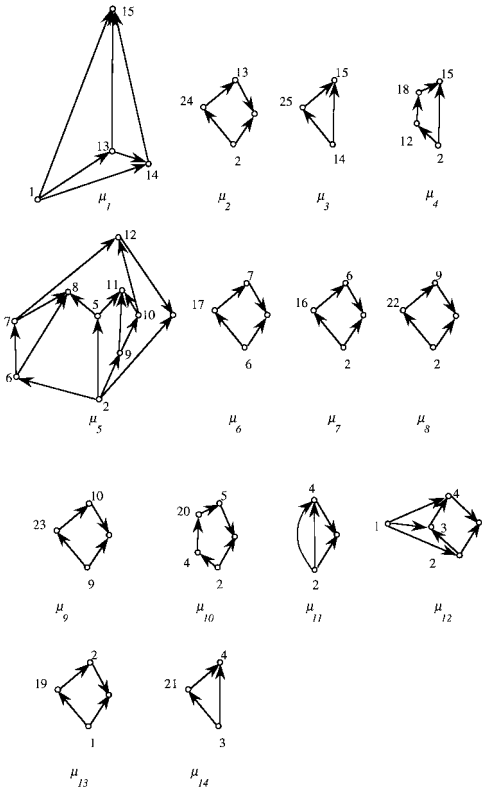


FIG. 17. *The sT-skeletons of the nodes of* $\mathcal{T}$.

(c) In Figure 19, the edges of the SPQR-tree $\mathcal{T}$ associated with unmarked virtual edges are oriented. For example, the tree-edge associated with the directed virtual edge (13, 14) of the sT-skeleton of μ_1 is oriented toward μ_1. Analogously, the tree-edge (μ_5, μ_{10}) associated with the directed-virtual-edge (2, 5) of sT-skeleton of μ_5 is directed toward μ_5.

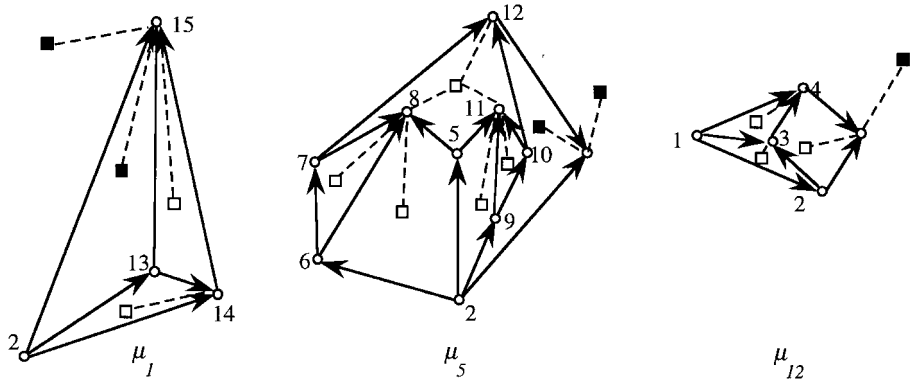


FIG. 18. *The face-sink graphs of the sT-skeletons of the* R*-nodes of* $\mathcal{T}$.

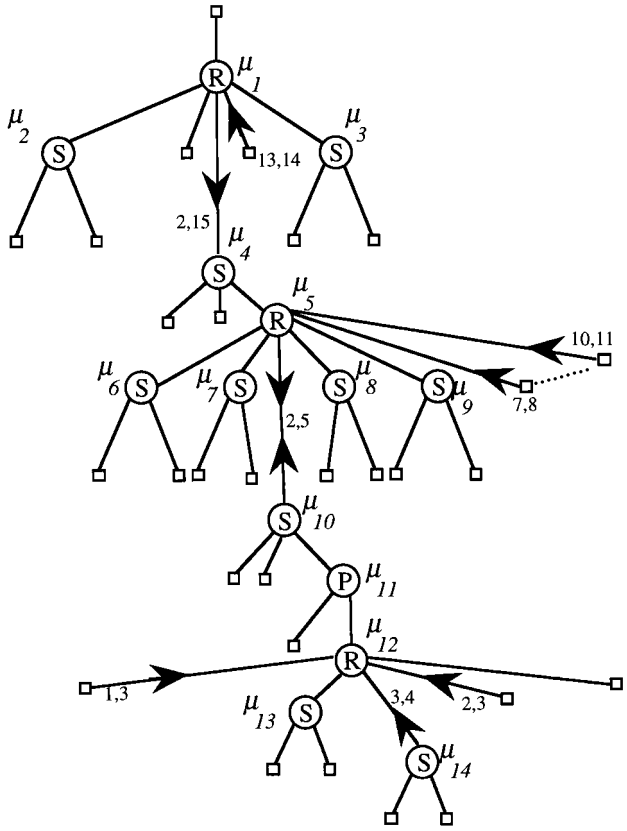


FIG. 19. *Partial orientation of* $\mathcal{T}$.

(d) Since the skeleton of μ_1 does not contain the source 1 of G', and the pertinent graph G'_4 associated with μ_4 contains the source 1, then the tree-edge μ_1, μ_4 is oriented toward μ_4. Analogously, the tree-edge μ_5, μ_{10} is oriented towards μ_{10}.

8. Since the tree-edge (μ_5, μ_{10}) is constrained to be oriented in both directions, it follows that it is not possible to find a rooting of $\mathcal{T}$ at a Q-node containing the source. Hence algorithm *Test* returns "not-upward-planar."

Example 2. In this example, we consider a graph H obtained from graph G of Figure 2a by contracting edge (12, 15).

1. In Figure 20 we show the expansion graph H' of H.

2. By a simple inspection it is possible to verify that H' is planar.

3. By a simple inspection it is possible to verify that H' is acyclic.

4. In Figure 21 we show the SPQR-tree $\mathcal{T}_H$ of H' and in Figure 22 the skeletons of the nodes of $\mathcal{T}_H$ (the skeletons of the Q-nodes are omitted).

5. In the same way as in step 5 of the previous example, all the endnodes of the virtual edges can be classified.

6. The sT-skeletons corresponding to the skeletons of Figure 22 are shown in Figure 23.

7. (a) In Figure 24 we show the face-sink graphs associated with the sT-skeletons of the R-nodes μ_1, μ_4, and μ_{11} (the vertices associated with the faces are represented by squares). It is easy to verify that they all satisfy the conditions of Theorem 1;

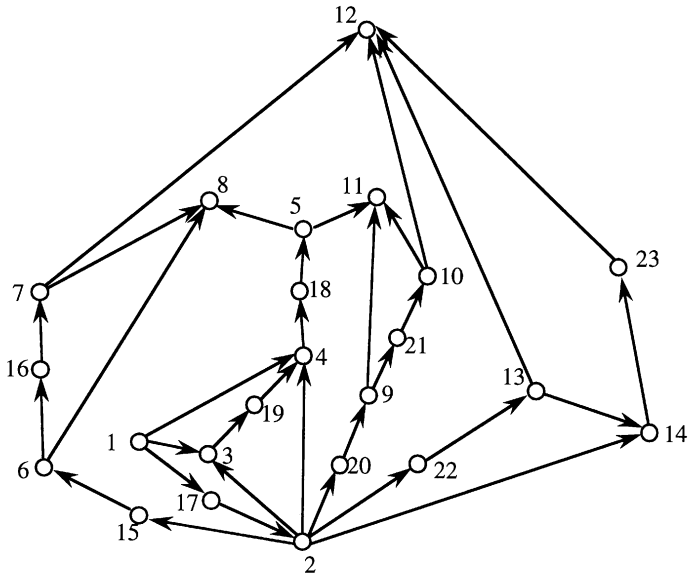


FIG. 20. *Expansion graph of graph H.*

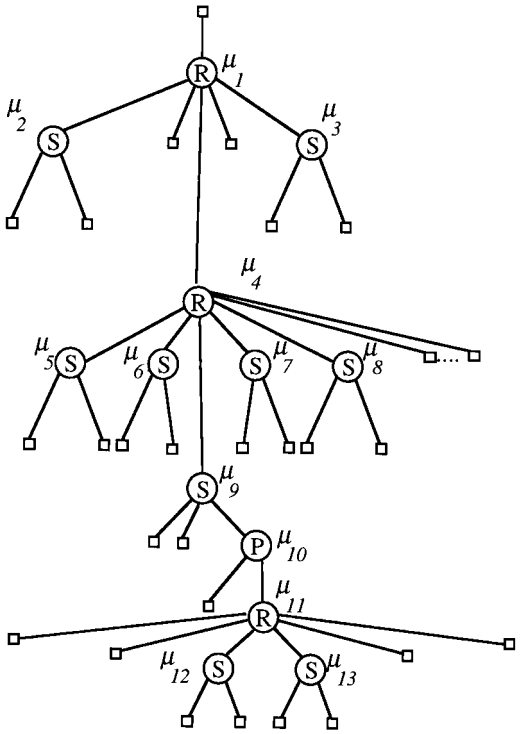


FIG. 21. *The SPQR-tree $\mathcal{T}_H$ of H'.*

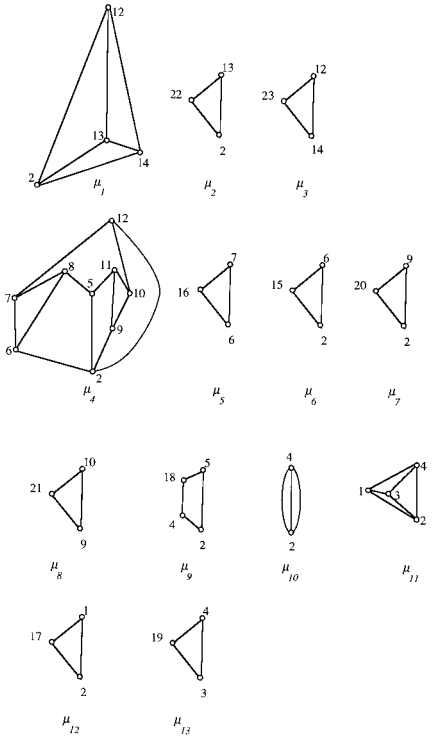


FIG. 22. *The skeletons of the nodes of* $\mathcal{T}_H$.

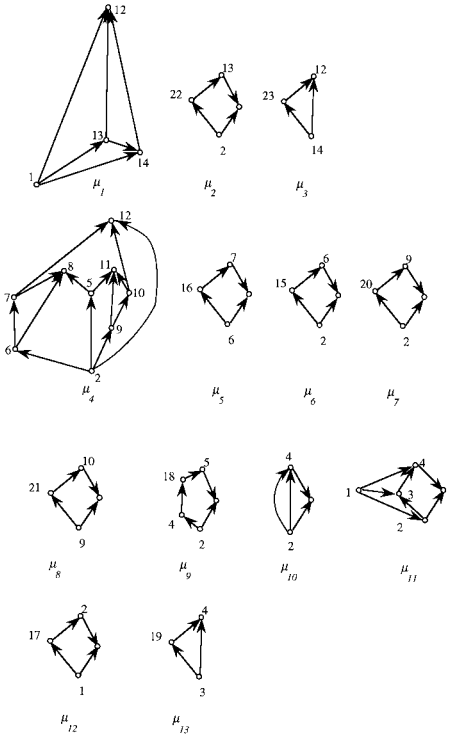


FIG. 23. *The* sT*-skeletons of the nodes of* $\mathcal{T}_H$.

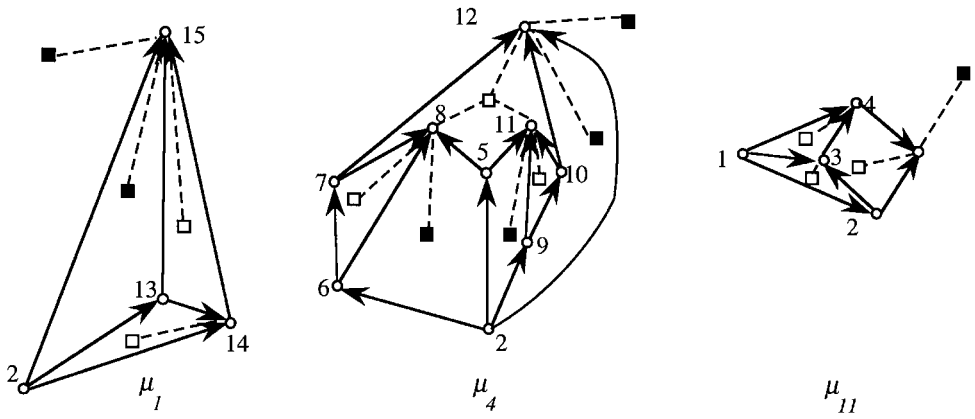


FIG. 24. *The face-sink graphs of the sT-skeletons of the* R*-nodes of* $\mathcal{T}_H$.

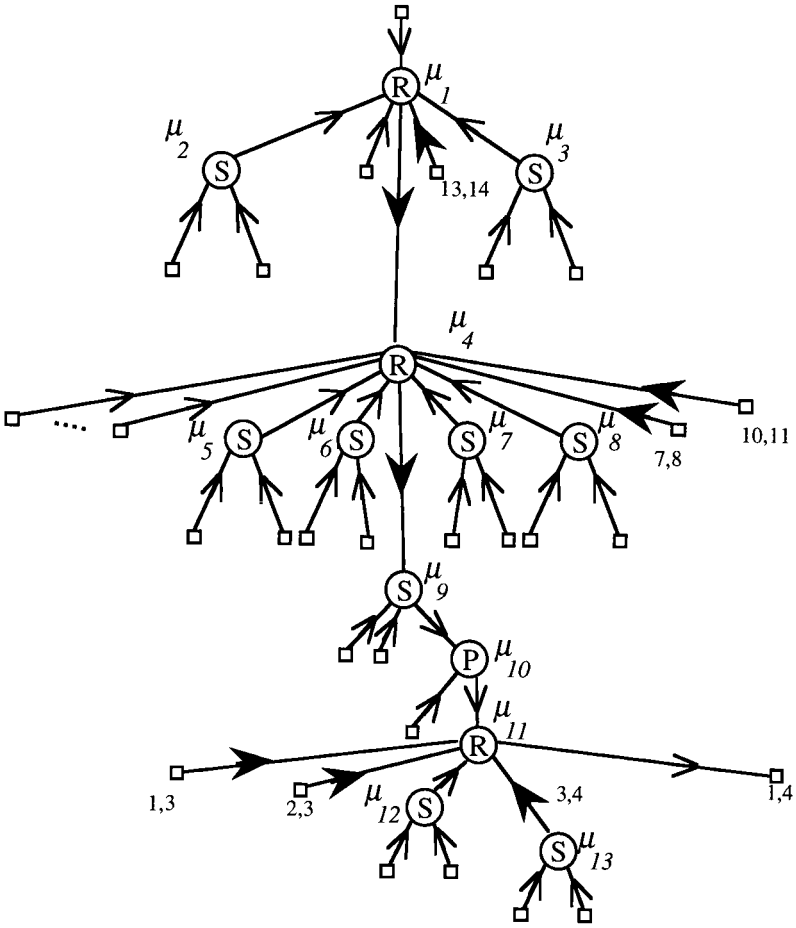


FIG. 25. *Orientation of* $\mathcal{T}_H$.

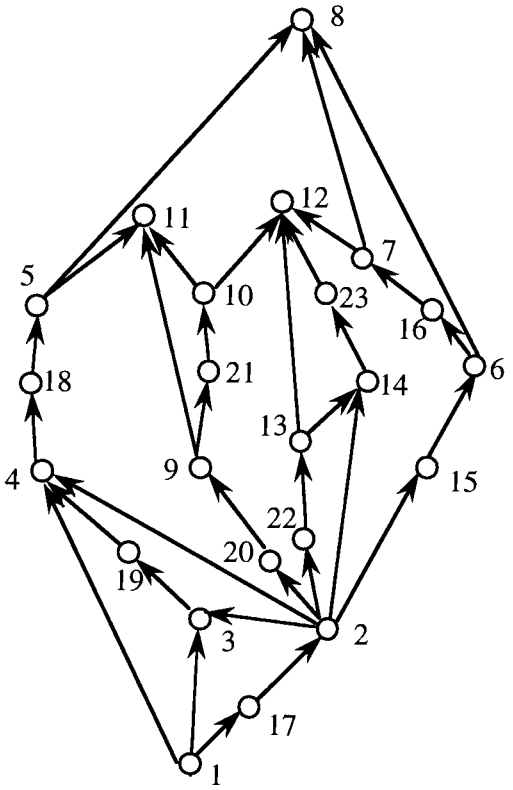


FIG. 26. *Upward drawing of H'.*

thus the algorithm *Embedded Test* will return "upward-planar" for every R-node. It also returns, for each R-node, the set of faces that can be external faces in an upward drawing of the associated sT-skeleton. In the figure, the nodes associated with these faces are indicated by black squares.

(b) By inspection of Figure 24, the unmarked edges are the following:

- μ_1: $\{(13, 14)\}$.
- μ_4: $\{(7, 8), (10, 11)\}$.
- μ_{11}: $\{(1, 3), (2, 3), (3, 4)\}$.

(c) In Figure 25, the edges of the SPQR-tree $\mathcal{T}_H$ associated with unmarked virtual edges are oriented; their orientations are represented by bold arrows.

(d) Since the skeleton of μ_1 does not contain the source 1 of H', and the pertinent graph H'_4 associated with μ_4 contains the source 1, then the tree-edge (μ_1, μ_4) is oriented toward μ_4. Analogously, the tree-edge (μ_4, μ_9) is oriented toward μ_9. Again, in Figure 25 these fixed orientations are represented by bold arrows.

8. By simple inspection, it is possible to find a rooting of $\mathcal{T}_H$ at the Q-node associated with the edge (1, 4) (see Figure 25). Hence algorithm *Test* returns "upward-planar."

An upward drawing of the expansion H' of graph H is shown in Figure 26.

REFERENCES

[1] P. Bertolazzi, R. F. Cohen, G. Di Battista, R. Tamassia, and I. G. Tollis, *How to draw a series-parallel digraph*, in Proc. 3rd Scand. Workshop Algorithm Theory, Lecture Notes in Comput. Sci. 621, Springer-Verlag, Berlin, 1992, pp. 272–283.

[2] P. Bertolazzi, R. F. Cohen, G. Di Battista, R. Tamassia, and I. G. Tollis, *How to draw a series-parallel digraph*, Internat. J. Comput. Geom. Appl., (4) 1994, pp. 385–402.

[3] P. Bertolazzi and G. Di Battista, *On upward drawing testing of triconnected digraphs*, in Proc. 7th Annual ACM Sympos. Comput. Geom., 1991, ACM, New York, pp. 272–280.

[4] P. Bertolazzi, G. Di Battista, G. Liotta, and C. Mannino, *Upward drawings of triconnected digraphs*, Algorithmica, 12 (1994), pp. 476–497.

[5] R. Cole and U. Vishkin, *Approximate and exact parallel scheduling with applications to list, tree, and graph problems*, in Proc. 27th IEEE Symp. on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1986, pp. 478–491.

[6] P. Crescenzi, G. Di Battista, and A. Piperno, *A note on optimal area algorithms for upward drawings of binary trees*, Comput. Geom. Theory Appl., 2 (1992), pp. 187–200.

[7] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Algorithms for drawing graphs: an annotated bibliography*, Comput. Geom. Theory Appl., 4 (1994), pp. 235–282.

[8] G. Di Battista, W. P. Liu, and I. Rival, *Bipartite graphs upward drawings and planarity*, Inform. Process. Lett., 36 (1990), pp. 317–322.

[9] G. Di Battista and R. Tamassia, *Algorithms for plane representations of acyclic digraphs*, Theoret. Comput. Sci., 61 (1988), pp. 175–198.

[10] G. Di Battista and R. Tamassia, *On-line graph algorithms with spqr-trees*, in Automata, Languages and Programming (Proc. 17th ICALP), Lecture Notes in Comput. Sci. 442, Springer-Verlag, New York, 1990, pp. 598–611.

[11] G. Di Battista and R. Tamassia, *On-line maintenance of triconnected components with SPQR-trees*, Algorithmica, 15 (1996), pp. 302–318.

[12] G. Di Battista, R. Tamassia, and I. G. Tollis, *Area requirement and symmetry display of planar upward drawings*, Discrete Comput. Geom., 7 (1992), pp. 381–401.

[13] P. Eades, T. Lin, and X. Lin, *Minimum size h-v drawings*, in Advanced Visual Interfaces (Proc. AVI '92), World Scientific Series in Computer Science 36, World Scientific, River Edge, NJ, 1992, pp. 386–394.

[14] D. Fussell, V. Ramachandran, and R. Thurimella, *Finding triconnected components by local replacements*, in Automata, Languages and Programming (Proc. 16th ICALP), Lecture Notes in Comput. Sci. 372, Springer-Verlag, New York, 1989, pp. 379–393.

[15] A. Garg, M. T. Goodrich, and R. Tamassia, *Area-efficient upward tree drawings*, in Proc. 9th Annual ACM Sympos. Comput. Geom., ACM, New York, 1993, pp. 359–368.

[16] A. Garg and R. Tamassia, *On the computational complexity of upward and rectilinear planarity testing*, in Proc. Graph Drawing '94, Lecture Notes in Comput. Sci. 894, Springer-Verlag, New York, 1995, pp. 286–297.

[17] J. Hopcroft and R. E. Tarjan, *Dividing a graph into triconnected components*, SIAM J. Comput., 2 (1973), pp. 135–158.

[18] J. Hopcroft and R. E. Tarjan, *Efficient planarity testing*, J. ACM, 21 (1974), pp. 549–568.

[19] M. D. Hutton and A. Lubiw, *Upward planar drawing of single source acyclic digraphs*, in Proc. 2nd ACM–SIAM Sympos. Discrete Algorithms, San Francisco, CA, SIAM, Philadelphia, 1991, pp. 203–211.

[20] M.-Y. Kao and G. E. Shannon, *Local reorientations, global order, and planar topology*, in Proc. 30th IEEE Symp. on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1989, pp. 286–296.

[21] D. Kelly, *Fundamentals of planar ordered sets*, Discrete Math., 63 (1987), pp. 197–216.

[22] D. Kelly and I. Rival, *Planar lattices*, Canad. J. Math., 27 (1975), pp. 636–665.

[23] A. Lempel, S. Even, and I. Cederbaum, *An algorithm for planarity testing of graphs*, in Theory of Graphs: Internat. Symposium, Rome, 1966, Gordon and Breach, New York, 1967, pp. 215–232.

[24] L. Lovasz and M. D. Pummer, *Matching Theory*, Ann. Discrete Math. 29, North–Holland, Amsterdam, 1986.

[25] A. Papakostas, *Upward planarity testing of outerplanar dags*, in Proc. Graph Drawing '94, Lecture Notes in Comput. Sci. 894, Springer-Verlag, New York, 1995, pp. 298–306.

[26] C. Platt, *Planar lattices and planar graphs*, J. Combin. Theory Ser. B, 21 (1976), pp. 30–39.

[27] V. Ramachandran and J. Reif, *Planarity Testing in Parallel*, Tech. Report TR-90-15, Department of Computer Science, University of Texas at Austin, 1990.

[28] V. Ramachandran and J. H. Reif, *An optimal parallel algorithm for graph planarity*, in

Proc. 30th Annual IEEE Sympos. Found. Comput. Sci., IEEE Computer Society Press, Los Alamitos, CA, 1989, pp. 282–293.

[29] E. Reingold and J. Tilford, *Tidier drawing of trees*, IEEE Trans. Software Eng., SE-7(1981), pp. 223–228.

[30] I. Rival, *Reading, drawing, and order*, in Algebras and Orders, I. G. Rosenberg and G. Sabidussi, eds., Kluwer Academic Publishers, Norwell, MA, 1993, pp. 359–404.

[31] K. J. Supowit and E. M. Reingold, *The complexity of drawing trees nicely*, Acta Inform., 18 (1983), pp. 377–392.

[32] R. Tamassia and J. S. Vitter, *Parallel transitive closure and point location in planar structures*, SIAM J. Comput., 20 (1991), pp. 708–725.

[33] C. Thomassen, *Planar acyclic oriented graphs*, Order, 5 (1989), pp. 349–361.

[34] W. T. Trotter and J. Moore, *The dimension of planar posets*, J. Combin. Theory Ser. B., 22 (1977), pp. 54–67.

SIAM J. COMPUT.
Vol. 27, No. 1, pp. 170–201, February 1998
© 1998 Society for Industrial and Applied Mathematics
008

LINEAR AND $O(n \log n)$ TIME MINIMUM-COST MATCHING ALGORITHMS FOR QUASI-CONVEX TOURS*

SAMUEL R. BUSS† AND PETER N. YIANILOS‡

Abstract. Let G be a complete, weighted, undirected, bipartite graph with n red nodes, n' blue nodes, and symmetric cost function $c(x, y)$. A maximum matching for G consists of $\min\{n, n'\}$ edges from distinct red nodes to distinct blue nodes. Our objective is to find a minimum-cost maximum matching, i.e., one for which the sum of the edge costs has minimal value. This is the weighted bipartite matching problem or, as it is sometimes called, the assignment problem.

We report a new and very fast algorithm for an abstract special case of this problem. Our first requirement is that the nodes of the graph are given as a "quasi-convex tour." This means that they are provided circularly ordered as $x_1, \ldots, x_N$, where $N = n + n'$, and that for any x_i, x_j, x_k, x_ℓ, not necessarily adjacent but in tour order, with x_i, x_j of one color and x_k, x_ℓ of the opposite color, the following inequality holds:

$$c(x_i, x_\ell) + c(x_j, x_k) \leq c(x_i, x_k) + c(x_j, x_\ell).$$

If $n = n'$, our algorithm then finds a minimum-cost matching in $O(N \log N)$ time. Given an additional condition of "weak analyticity," the time complexity is reduced to $O(N)$. In both cases only linear space is required. In the special case where the circular ordering is a line-like ordering, these results apply even if $n \neq n'$.

Our algorithm is conceptually elegant, straightforward to implement, and free of large hidden constants. As such we expect that it may be of practical value in several problem areas.

Many natural graphs satisfy the quasi-convexity condition. These include graphs which lie on a line or circle with the canonical tour ordering, and costs given by any concave-down function of arclength — or graphs whose nodes lie on an arbitrary convex planar figure with costs provided by Euclidean distance.

The weak-analyticity condition applies to points lying on a circle with costs given by Euclidean distance, and we thus obtain the first linear-time algorithm for the minimum-cost matching problem in this setting (and also where costs are given by the L_1 or L_∞ metrics).

Given two symbol strings over the same alphabet, we may imagine one to be red and the other blue and use our algorithms to compute string distances. In this formulation, the strings are embedded in the real line and multiple independent assignment problems are solved, one for each distinct alphabet symbol.

While these examples are somewhat geometrical, it is important to remember that our conditions are purely abstract; hence, our algorithms may find application to problems in which no direct connection to geometry is evident.

Key words. assignment problem, bipartite weighted matching, computational geometry, concave penalty function, convexity, linear time, Monge property, quadrangle inequality, string comparison

AMS subject classifications. 05C70, 05C85, 05C90, 52A37, 68Q20, 68R10, 68U15, 90C27

PII. S0097539794267243

1. Introduction.

The above abstract gives a short overview of the contents of the paper, and we shall give an in-depth discussion of our definitions, results, and algorithm below. However, we first give a quick review of prior related work on

*Received by the editors May 9, 1994; accepted for publication (in revised form) January 7, 1996. A preliminary version of this paper appeared in the *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, Arlington, VA, SIAM, Philadelphia, 1994, pp. 65–76.

http://www.siam.org/journals/sicomp/27-1/26724.html

†Department of Mathematics, University of California, San Diego, La Jolla, CA 92092-0112 (sbuss@ucsd.edu). The research of this author was supported in part by NSF grants DMS-9205181 and DMS-9503247.

‡NEC Research Institute, 4 Independence Way, Princeton, NJ 08540 and Department of Computer Science, Princeton University, 35 Olden Street, Princeton, NJ 08544-2087 (pny@research.nj.nec.com).

matching. We shall consider graphs G which have N nodes; the nodes are partitioned into a set of n red nodes and n' blue nodes with $N = n + n'$. G is *balanced* if it has equal numbers of red and blue nodes. There is a symmetric cost function $c(x, y)$, which gives the cost of an edge from node x to node y, with x and y of distinct colors. A matching is a set of edges with no endpoints in common that match all the nodes of one color with the same number of nodes of the opposite color. The cost of a matching is the sum of the costs of its edges. The problem of finding a minimal-cost matching for a general bipartite graph is known to have an $O(N^3)$ time algorithm (see Lawler [18] for this and other background on matching), and for graphs with nodes in the plane with the Euclidean distance as cost function, there is a $O(N^{2.5} \log N)$ time algorithm due to Vaidya [22].

The minimum-cost matching problem is substantially easier in the case where the nodes are in line-like order or are circularly ordered. The simplest versions of line-like/circular orderings are where the points lie on a line or lie on a curve homeomorphic to a circle, and the cost $c(x, y)$ of an edge between x and y is equal to the shortest arclength distance between the nodes. The matching problem for this arclength cost function has been studied by Karp and Li [14], Aggarwal et al. [1], Werman et al. [23], and others, and is the "skis and skiers" problem of Lawler [18]. Karp and Li have given linear time algorithms for this matching problem; Aggarwal et al. have generalized the linear time algorithm to the transportation problem.

A more general version of the matching problem for graphs in line-like order has been studied by Gilmore and Gomory [10] (see [18]). In this version, the cost of an edge from a red node x forward to a blue node y is defined to equal $\int_x^y f$, and from a blue node x forward to a red node y to equal $\int_x^y g$, for some functions f and g. This matching problem has a linear time algorithm provided $f + g \geq 0$.

Another version of the matching problem for line-like graphs is considered by Aggarwal et al. [1]; they use graphs which satisfy a "Monge" property which states that the inequality (1.1) below holds except with the inequality sign's direction reversed. They give a linear time algorithm for the matching problem for (unbalanced) Monge graphs.

In the prior work most closely related to this paper, Marcotte and Suri [20] consider the matching problem for a circularly ordered, balanced tour in which the nodes are the vertices of a convex polygon and the cost function is equal to Euclidean distance. This matching problem is substantially more complicated than the comparatively simple "skis and skiers" type problems; nonetheless, Marcotte and Suri give an $O(N \log N)$ time algorithm which solves this minimum-cost matching problem. For the case where the nodes are the vertices of a simple polygon and the cost function is equal to the shortest Euclidean distance *inside* the polygon, they give an $O(N \log^2 N)$ time algorithm.

The main results of this paper apply to all of the above matching problems on circularly ordered or line-like tours, with the sole exception of unbalanced, Monge graphs. For the "skis and skiers" and the problems of Gilmore and Gomory, Theorem 1.9 gives new linear time algorithms that find minimum-cost matchings which are different than the traditional minimum-cost matchings (and our algorithms are more complicated than is necessary for these simple problems). Our algorithms subsume those of Marcotte and Suri and give some substantial improvements. First, with the weak analyticity condition, we have linear time algorithms for many important cases, whereas Marcotte and Suri's algorithm takes $O(N \log N)$ time. Second, our assumption of quasi convexity is considerably more general than their planar geometrical

setting and allows diverse applications. Third, our algorithms are conceptually simpler than the divide-and-conquer methods used by Marcotte and Suri, and we expect that our algorithms are easier to implement.

All of our algorithms have been implemented as reported in [5]; a brief overview of this implementation is given in section 3.4.

We list some sample applications of our algorithms in the examples numbered 1–8 below. One example of a matching problem solution is shown in Figure 1.1. For this figure, a 74-node bipartite graph was chosen with nodes on the unit circle. For this matching problem, the cost of an edge is equal to the Euclidean distance between its endpoints. The edges shown form a minimum-cost matching.

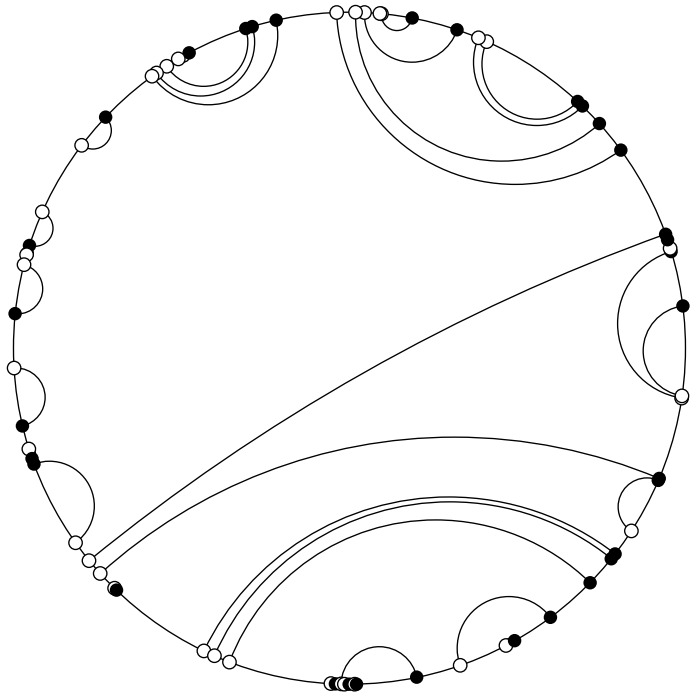


FIG. 1.1. *The minimum-cost matching for a* 74*-node graph on the circle with Euclidean distance as the cost function.*

Our quasi-convex property is equivalent to the "inverse quadrangle inequality" used, for instance, by [8] but is weaker than the similar "inverse Monge property" of [4]. In fact, we show below that any Monge matching problem may be trivially transformed into a quasi-convex matching problem, but not vice-versa.

Dynamic programming problems based on cost functions which satisfy the (inverse) quadrangle inequality, and some closely related matrix-search problems, have been studied by many authors including [2, 3, 4, 7, 8, 9, 12, 15, 16, 17, 19, 24, 25]. However, we have discovered no direct connection between our quasi-convex matching problem and the problems solved by these authors.

The notion of a *Monge array* [13] is related to that of quasi convexity, but the Monge condition is stronger (i.e., quasi convexity is strictly more general). Because of the similarity between the definitions of both properties, we take the time to illustrate this point in detail. To understand the Monge property in our bipartite setting, imagine the cost function to be an array, and impose the restriction that its first

argument select a red point and the second a blue point. The array is Monge provided that for all i, j, k, ℓ satisfying $1 \le i < j \le n$ and $1 \le k < \ell \le n'$, we have

$$c(R_i, B_k) + c(R_j, B_\ell) \le c(R_i, B_\ell) + c(R_j, B_k).$$

Now given a graph with a Monge cost array, we convert it (in linear time) to a quasi-convex tour by simply visiting the red vertices first, in Monge order, followed by the blue vertices, in reverse Monge order. The quasi-convexity inequality is then an immediate consequence of the Monge property and our reverse ordering of the blue vertices. This reversal is necessary because the sense of the Monge inequality is opposite that of quasi convexity.

However, not every quasi-convex tour can be rearranged to form a Monge array. We will now exhibit such a quasi-convex tour. Its nodes lie along the real line, and costs are given by the square root of internode distance; tour order is from left to right. Given a subtour of the form $R_iR_jB_aB_b$, then it is easily shown that in any Monge reordering, $B_a \prec B_b$ iff $R_j \prec R_i$. Similarly, given a tour of the form $R_jB_aB_bR_i$, $B_a \prec B_b$ iff $R_j \prec R_i$. Our counterexample then consists of any tour having a subtour of the form $RBBRRBB$. To understand why, we attach subscripts resulting in $R_iB_aB_bR_jR_kB_cB_d$ and proceed to apply the two rules above to get the implications

$$B_a \prec B_b \implies R_i \prec R_j \implies B_d \prec B_c \implies R_j \prec R_k.$$

Also,

$$B_a \prec B_b \implies R_k \prec R_j,$$

which is a contradiction. Symmetrically the same conclusion is reached if one begins instead with $B_b \prec B_a$, whence no Monge rearrangement exists.

We now give the definitions necessary to state the main results of this paper. We think of the nodes of the graph G as being either a line-like or circular tour of the graph; in the case of a circular tour, we think of the node x_1 as following again after x_N.

DEFINITION 1.1. *A sequence of nodes $x_{i_1}, x_{i_2}, \ldots, x_{i_\ell}$ are* in input order *if and only if $i_1 < i_2 < \cdots < i_\ell$. The nodes are defined to be* in tour order *if and only if there exists a k such that the sequence $x_{i_k}, \ldots, x_{i_\ell}, x_{i_1}, \ldots, x_{i_{k-1}}$ is in input order.*

DEFINITION 1.2. *The nodes $x_1, \ldots, x_N$ form a* quasi-convex *tour if and only if, whenever x_i, x_j, x_k, x_ℓ are in tour order, with x_i and x_j of one color and x_k and x_ℓ of the other color, then*

$$c(x_i, x_\ell) - c(x_i, x_k) \le c(x_j, x_\ell) - c(x_j, x_k). \tag{1.1}$$

Reordering terms in (1.1) gives

$$c(x_i, x_\ell) + c(x_j, x_k) \;\le\; c(x_i, x_k) + c(x_j, x_\ell).$$

To give a geometric intuition to quasi convexity, note that when x_i, x_j, x_k, x_ℓ are the vertices of a quadrilateral, the inequality states that the sum of the lengths of diagonals is greater than or equal to the sum of the lengths of two of the sides.

DEFINITION 1.3. *The tour $x_1, \ldots, x_N$ of G is* line-like *if and only if the following holds: For all $i < j < k$, we have*

$$c(x_i, x_j) \le c(x_i, x_k)$$

if x_i is of opposite color from x_j and x_k, and we have

$$c(x_i, x_k) \geq c(x_j, x_k)$$

if x_k is of opposite color from x_i and x_j.

The property of quasi convexity is defined independently of the starting point of the tour; i.e., the nodes of the tour can be "rotated" without affecting quasi convexity. Obviously, the definition of line-like tours is sensitive to the choice of starting point of the tour.

Our main theorems give either $O(N \log N)$ or $O(N)$ time algorithms for all of the following examples, with the exception of example 7:

1. Let the nodes $x_1, \ldots, x_N$ be sequentially ordered points on a line (e.g., they are real numbers indicating points on the x-axis), and let $||x_j - x_i||$ be the Euclidean distance from x_i to x_j. Let f be any concave-down function, so $f''(x) \leq 0$ for all x. If the cost function is defined by

$$c(x_i, x_j) \;=\; f(||x_j - x_i||), \tag{1.2}$$

then $x_1, \ldots, x_N$ form a quasi-convex tour. Prior work for examples 1 and 2 gave linear time matching algorithms *only* for the case where $f(x)$ is a linear function [14, 1].

2. Now let the points $x_1, \ldots, x_N$ lie on a smooth curve C which is homeomorphic to a circle, with the points listed in, say, counterclockwise order. And let $||x_j - x_i||$ equal the shortest arclength along C from x_i to x_j. Again let $f(x)$ be any concave down function. With the cost function given by equation (1.2), the nodes $x_1, \ldots, x_N$ form a quasi-convex tour.

3. Suppose $x_1, \ldots, x_N$ lie, in that order, on a circle. Let $c(x_i, x_j)$ equal the Euclidean distance from x_i to x_j. Since Euclidean distance is a concave-down function of the circular arclength, this is a special case of example 2 and the nodes form a quasi-convex tour. In this case, the weak analyticity condition always holds and Main Theorem 1.9 gives an $O(N)$ time algorithm. The best prior algorithm was $O(N \log N)$ time [20].

4. More generally, if $x_1, \ldots, x_N$ are the vertices of a convex polygon listed in, say, counterclockwise order, and if the cost function is equal to Euclidean distance, then the nodes form a quasi-convex tour. The prior algorithm for this case was $O(N \log N)$ time [20] and our algorithms are either $O(N)$ or $O(N \log N)$ time depending on whether the weak analyticity condition holds.

5. Some nonconvex polygons also have vertices which form a quasi-convex tour. For example, in a polygon shaped as in Figure 1.2, the vertices A, B, C, D will form a quasi-convex tour, provided the angle θ not too large. (This is why we use "quasi-convex" instead of "convex" to describe tours which satisfy equation (1.1).)

6. Examples 4 and 5 are also quasi-convex under other distance metrics such as the L_1 and L_∞ metrics.

7. Marcotte and Suri consider graphs where the nodes are the vertices of a simple polygon and the cost function is equal to the length of the shortest connecting path inside the polygon. The nodes of such a polygon form a quasi-convex tour. The prior algorithm and the algorithm of this paper are $O(N \log^2 N)$ time for this example, since the cost function requires $O(\log N)$ time to compute.

8. In string matching algorithms, the cost of shifting a character's position is specified as a function of the distance shifted. The authors have worked in the past on string matching algorithms [26, 27] in which the cost function is a linear function

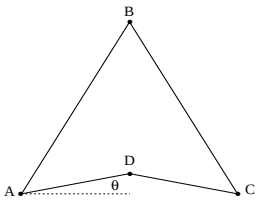


FIG. 1.2. *A quasi-convex polygon which is not geometrically convex.*

of distance. These prior algorithms have been quite successfully used in commercial applications, especially natural language search, and we expect that the use of a concave-down distance function will significantly improve the matching quality. As we discuss in section 5, the setting of example 1 above is precisely what is needed to allow (near) linear time string matching algorithms with concave-down cost functions. A number of authors, including [7, 8], have studied concave-down cost functions for string matching; their string matching algorithms are based on least-edit-distance and, in this regard, are quite different from ours. Least-edit-distance string matching algorithms are widely used because they provide rich and flexible string comparison functions; on the other hand, the best general algorithms for computing least-edit-distance require $O(N^2)$ time (see [21]). Our string matching algorithms are not as flexible but can be tailored to work well for many applications; they have the advantage of being linear time computable.

MAIN THEOREM 1.4.

(i) *There is an $O(N \log N)$ time algorithm for the minimum-cost matching problem for line-like quasi-convex tours.*

(ii) *There is an $O(N \log N)$ time algorithm for the minimum-cost matching problem for balanced quasi-convex tours.*

Remark. The running times of the algorithms are given in terms of the number N of nodes, even though the input size may in some cases need to be $\Omega(N^2)$ to fully specify the values of the cost function. However, in all the examples above, the input size is $O(N)$ since the cost function is specified by the nodes' positions on a line, on a curve, or in the plane. In any event, our runtime analysis assumes that any value $c(x_i, x_j)$ of the cost function can be computed in constant time. If this is not the case, then the runtimes are to be multiplied by the time needed to compute a value of the cost function; this is the situation in example 7 above.

We next define a "weak analyticity" condition which will allow even faster algorithms.

DEFINITION 1.5. *Suppose that x_i and x_j are red (blue) nodes, that $\delta \geq 0$, and that there is a blue (respectively, red) node x_k such that*

$$c(x_i, x_k) - c(x_j, x_k) < \delta.$$

The δ-crossover point *of x_i and x_j is defined to be the first such x_k, where "first" means in tour order starting from x_j and ending at x_i. If no such x_k exists, then the δ-crossover point does not exist.*

It is not hard to see that the property of quasi convexity implies that, if the δ-crossover point x_k exists, then $c(x_i, x_\ell) - c(x_j, x_\ell) \geq \delta$ whenever x_i, x_j, x_ℓ, x_k are in tour order and $c(x_i, x_\ell) - c(x_j, x_\ell) < \delta$ whenever x_i, x_j, x_k, x_ℓ are in tour order. Thus binary search provides an $O(\log N)$ time procedure which, given x_i, x_j, and δ, will determine if x_k exists and, if so, which node is x_k. This is the approach taken in

the algorithms of Theorem 1.4 and is the source of the $\log N$ factor in the runtime. However, in some cases, x_k can be found in constant time and we define the following.

DEFINITION 1.6. *A quasi-convex tour satisfies the* strong analyticity condition *provided there is a constant-time algorithm which can determine if the δ-crossover point of x_i and x_j exists and, if so, can determine which node it is.*

A quasi-convex tour satisfies the analyticity condition *provided there is a constant-time algorithm which can answer the following question (as a function of similarly colored nodes x_i, x_j, x_k in tour order and of $\epsilon, \delta > 0$, where the δ-crossover of x_i and x_j is known to exist):*

> *"Do x_j and x_k have an ϵ-crossover point which either equals or precedes in tour order the δ-crossover point of x_i and x_j?"*

Even the analyticity condition is too strong to be satisfied in many situations, so we also define a "weak analyticity condition" as follows.

DEFINITION 1.7. *Let x be a node and y and z be denotations of nodes. We write $y \prec_x z$ to denote that either* (i) *y and z exist and are distinct and y precedes z in the tour order beginning at x, or* (ii) *y exists and z does not.*

A relative crossover procedure *is a procedure Ω such that, given ϵ, δ, x_i, x_j, and x_k as input, and letting y be the δ-crossover of x_i and x_j, and z be the ϵ-crossover of x_j and x_k, then*

(i) *If $y \prec_{x_j} z$, then Ω outputs "Yes."*
(ii) *If $z \prec_{x_j} y$, then Ω outputs "No."*
(iii) *Otherwise Ω may output either answer.*

Note that Ω is not required to determine y and z. The difference between weak analyticity and ordinary analyticity is that when condition (iii) *holds, Ω may output either answer.*

DEFINITION 1.8. *The* weak analyticity condition *is said to hold provided there is a constant-time relative crossover procedure.*

Clearly the strong analyticity condition implies the analyticity condition, which in turn implies the weak analyticity condition. In most applications, we do not have the analyticity or strong analyticity conditions, but the weak analyticity condition does hold in many natural situations. In particular, examples 1, 2, 3, and 4 do satisfy the weak analyticity condition provided that the concave-down function is sufficiently natural. Consider, for instance, example 1 with the concave-down function $f(x) = x$, $f(x) = \sqrt{x}$, or $f(x) = \log x$, etc. For example 1, the input nodes $x_1, \ldots, x_N$ are given with a sequence of real numbers $r_1 \leq r_2 \leq \cdots \leq r_N$ which are the positions of the nodes on the real line. Given nodes x_i, x_j and $\delta > 0$, the first possible position for the δ-crossover of x_i and x_j can be found by solving the equation $f(y - r_i) = \delta + f(y - r_j)$ for y; since we assume that arithmetic operations take constant time, the solution y can be found in constant time. Note that y is only the *theoretical* crossover point; the actual crossover is the first node x_k such that $y \leq r_k$. Unfortunately, even after y is known, it will not be possible to determine x_k in constant time unless some additional information is given about the distribution of the nodes on the real line. Thus, the analyticity condition and strong analyticity conditions do not hold in general for example 1. The reason the analyticity condition does not hold is that, if the theoretical ϵ-crossover point occurs after the theoretical δ-crossover point, then the analyticity algorithm must output "No" if there is a node after the theoretical δ-crossover point and before or at the theoretical ϵ-crossover point, and must output "Yes" otherwise (because in the latter case the two actual crossover points coincide). Unfortunately, there is no general way to decide this in constant time, so the analyticity condition

is false. However, the weak analyticity condition does hold, since the function Ω may operate by computing the theoretical δ-crossover of x_i and x_j and the theoretical ϵ-crossover of x_j and x_k and outputting "Yes" if the former is less than the latter.

For similar reasons, example 3 satisfies the weak analyticity condition; in this case, since the nodes lie on a circle and the cost function is Euclidean distance, the theoretical crossover position is computed (in constant time) as the intersection of a hyperbola and the circle. Likewise, the weak analyticity condition also holds for example 2 if the concave-down function is sufficiently nice, and it holds for example 6, where nodes lie on a circle under the L_1 and L_∞ metrics. Example 4, where the nodes form the vertices of a convex polygon, does not seem to satisfy the weak analyticity condition in general; however, some important special cases do. For example, if the vertices of the convex polygon are known to lie on a polygon with a bounded number of sides, on an oval, or on a branch of a hyperbola, then the weak analyticity condition does hold.

The analyticity condition has been implicitly used by Hirschberg and Larmore [12] who defined a *Bridge* function which is similar to our Ω function. They give a special case in which *Bridge* is constant-time computable and thus the analyticity condition holds. Later, Galil and Giancarlo [8] defined a "closest zero property" which is equivalent to our strong analyticity condition.[1] As we illustrated above, the analyticity and strong analyticity conditions rarely hold. Thus it is interesting to note that the algorithms of Hirschberg and Larmore and of Galil and Giancarlo will still work, with only minor modifications, if only the weak analyticity condition holds.

Our second main theorem implies that these examples which satisfy the weak analyticity condition have linear time algorithms for minimum-cost matching.

MAIN THEOREM 1.9.

(i) *There is an $O(N)$ time algorithm for the minimum-cost matching problem for line-like quasi-convex tours which satisfy the weak analyticity condition.*

(ii) *There is an $O(N)$ time algorithm for the minimum-cost matching problem for balanced quasi-convex tours which satisfy the weak analyticity condition.*

Remark. In order to achieve the linear time algorithms, it is necessary that nodes of the graph be input in their tour order. This assumption is necessary, since without it, it is possible to give a linear time reduction of sorting to the matching problem for line-like tours.

Our main theorems also apply to minimum-cost matchings for some nonbipartite quasi-convex tours. If a nonbipartite graph G has N nodes and cost function c, then a matching for G is a set of $\lfloor \frac{1}{2}N \rfloor$ edges with all endpoints distinct. Part (i) of Main Theorems 1.4 and 1.9 hold also for nonbipartite graphs which are line-like quasi-convex tours. And part (ii) of Main Theorems 1.4 and 1.9 hold also for nonbipartite graphs which are quasi-convex tours with an even number of nodes. The nonbipartite cases are discussed in section 4; the algorithms are simple modifications of the algorithms for the bipartite tours.

It is apparent that our algorithms can be parallelized, but we have not investigated the precise runtime and processor count that is needed for a parallel implementation. He [11] has given a PRAM implementation of Marcotte and Suri's algorithm which uses N processors and $O(\log^2 N)$ time and it is clear that our algorithm can be

[1] The definition of the "closest zero property" is misstated in [8]; it should be defined as saying that it is possible to find the first r such that $w(l,r) - w(k,r) - a \leq 0$ (note their w corresponds to our cost function c, and a is a real). However, their algorithm explicitly uses the correct definition of "closest zero property" (see their Fact 2).

computed with the same number of processors with the same time bounds using He's methods.

Our algorithms apply to unbalanced tours only if they are line-like. This is because in the line-like case the leveling process, described in the next section, induced by choosing the first node as starting point, is guaranteed to decompose the problem into alternating color subproblems, which may be independently solved and reassembled to produce an overall solution. Now, some of these subproblems may be unbalanced, but again using the line-like property, we are able to force balance by adding a dummy node when necessary. These then are two different uses of the line-like property.

In balanced, unimodal tours[2] such as the circle, the leveling concept of section 2 holds in a weaker form. However, we have been unable to extend our results to the unbalanced unimodal case. As an example of the difficulty of this, consider the highly eccentric ellipse of Figure 1.3; the bipartite tour containing its four nodes is unbalanced and is neither line-like nor unimodal. Notice that no starting point induces a leveling which places R_2 and B_1 at the same level, despite the fact that the minimum-cost matching consists of an edge between them. The path to extending our methods to such cases is therefore less clear.

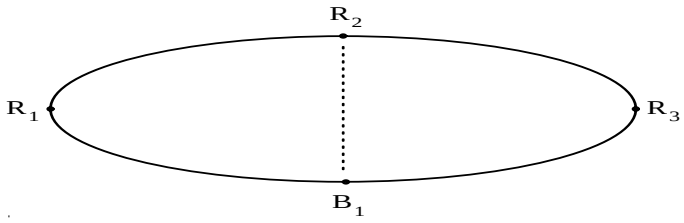


FIG. 1.3. *A bipartite, unbalanced, unimodal tour for which no leveling process works.*

2. Reductions and lemmas.

2.1. Reduction to tours of alternating colors. The first step to giving our minimum-cost matching algorithms is to reduce to the special case of tours in which the colors of the nodes alternate. In other words, we will be able to assume w.l.o.g. that $x_1, x_3, x_5, \ldots$, are red and that $x_2, x_4, x_6, \ldots$, are blue.

DEFINITION 2.1. *Let x_i and x_j be nodes. We write $[x_i, x_j]$ to denote the sequence of nodes obtained by starting with x_i and advancing in tour order to x_j. We write $(x_i, x_j]$, $[x_i, x_j)$, and (x_i, x_j) for this sequence minus the starting node, the ending node, or both.*

If x is a node, let $d(x)$ denote the number of red nodes in $[x_1, x)$ minus the number of blue nodes in $[x_1, x)$. The level *of x, $level(x)$, is equal to $d(x)$ if x is blue and is equal to $d(x) + 1$ if x is red. We write $x \sim y$ to mean that $level(x) = level(y)$; obviously, $\sim$ is an equivalence relation. It is easy to see that if y is the first node after x in input order such that $x \sim y$, then x and y are of opposite colors. Also, if $x \sim y$ and x, y are in input order and are of opposite colors, then (x, y) contains equal number of red and blue nodes. For balanced tours, the $\sim$-equivalence relation is invariant under circular rotation of the nodes in the tour.*

Given a matching on the nodes of a graph, we write $x_i \leftrightarrow x_j$ to indicate the presence of an edge between x_i and x_j in the matching. We say that x_j immediately

[2] In unimodal tours, the cost function from any node rises and then falls as the tour is traversed.

follows x_i *in tour order if* $j = i + 1$ *or if* $i = N$ *and* $j = 1$. *Two nodes* x_i *and* x_j *are* adjacent *if and only if one of them immediately follows the other. An edge* $x_i \leftrightarrow x_j$ *is called a* jumper *if* x_i *and* x_j *are not adjacent. Two jumpers are said to* cross *if they are of the form* $x_i \leftrightarrow x_k$ *and* $x_j \leftrightarrow x_\ell$ *with* x_i, x_j, x_k, x_ℓ *in tour order.*

LEMMA 2.2. *Let* G *be either a line-like quasi-convex tour or a balanced quasi-convex tour. Then* G *has a minimum-cost matching in which every edge* $x_i \leftrightarrow x_j$ *satisfies* $x_i \sim x_j$. *In other words, some minimum-cost matching for* G *can be obtained as a union of minimum-cost matchings on the* $\sim$*-equivalence classes of* G.

To prove Lemma 2.2 we use the following lemma.

LEMMA 2.3. *G has a minimum-cost matching in which no jumpers cross.*

Sketch of proof. If a minimum-cost matching does have a pair of jumpers which cross, the quasi-convexity property allows them to be "uncrossed" without increasing the total cost. Repeatedly uncrossing jumpers will eventually yield a minimum-cost matching with no crossing jumpers. (See Lemma 1 of [1] for a detailed proof of this.)

Lemma 2.2 is proved by noting that a minimum-cost matching with no crossing jumpers must respect the $\sim$-equivalence classes. This is because, if a jumper $x_i \leftrightarrow x_j$ is in a crossing-free matching with $i < j$, then the nodes in the interval (x_i, x_j) must be matched with each other and thus (x_i, x_j) must have equal numbers of red and blue nodes. In the unbalanced, line-like case, this also depends on the fact that, w.l.o.g., there is no jumper which crosses an unmatched node (this is an immediate consequence of the line-like condition). □

By Lemma 2.2, in order to find a minimum-cost matching, it suffices to extract the $\sim$-equivalence classes and find minimum-cost matchings for each equivalence class independently. It is an easy matter to extract the $\sim$-equivalence classes in linear time by using straightforward counting. Each equivalence class consists of an alternating color subtour; in the balanced case, there are an even number of nodes in each equivalence class, and in the line-like condition case, there may be an even or odd number of nodes. Thus, to give (near) linear time algorithms for finding matchings, it will suffice to restrict our attention to tours in which the nodes are of alternating colors.

In view of Lemma 2.3, we may restrict our attention to matchings which contain no crossing jumpers. Such a matching will be called *crossing-free*.

Finally, we can assume w.l.o.g. that the tour is balanced. To see why we can assume this, suppose that $x_1, \ldots, x_N$ is an unbalanced, line-like tour of alternating colors. This means that x_1 and x_N are the same color, say red. We can add a new node x_{N+1} to the end of the tour, label it blue, and let $c(x_i, x_{N+1}) = 0$ for all red x_i. These $N + 1$ nodes no longer form a line-like tour; however, they do form a balanced quasi-convex tour. Solving the matching problem for the $N + 1$ nodes immediately gives a solution to the matching problem on the original N nodes.

2.2. Some important lemmas. Since we are now working only with balanced quasi-convex tours of alternating colors, we shall often change the names of the nodes to $R_1, B_1, \ldots, R_M, B_M$; so R_i and B_j refer to the ith red node and the jth blue node in the tour, respectively. (So x_{2i-1} is the same as R_i and x_{2i} is the same as B_i.) Note that this means $N = 2M$. To simplify notation, we define

$$c_i \;=\; c(R_i, B_i) \qquad \text{and} \qquad c'_i \;=\; c(B_i, R_{i+1}).$$

A *greedy* matching is a matching which contains no jumpers, i.e., every node is matched to an adjacent node. There are two greedy matchings, namely, the one containing all edges $R_i \leftrightarrow B_i$ and the one containing all edges $B_{i-1} \leftrightarrow R_i$ and the edge $B_M \leftrightarrow R_1$. For x_i and x_j nodes of opposite color, a matching σ is said to

be *greedy on* (x_i, x_j) provided it contains as a submatching the unique matching of adjacent nodes contained in the interval (x_i, x_j). We similarly define the notion of σ being greedy on a balanced interval I, where I is one of the intervals $[x_i, x_j)$, $[x_i, x_j]$, or $(x_i, x_j]$, but with the additional provisos that $x_i \leftrightarrow x_{i+1}$ is in σ in the first two cases and that $x_{j-1} \leftrightarrow x_j$ is in σ in the second two cases.[3]

The notation $[R_i, B_j]$ has already been defined. In addition, the notation $[i, j]$ denotes the interval of integers $i, i+1, \ldots, j$ if $i < j$, or the (circular) interval $i, i+1, \ldots, M, 1, 2, \ldots, j$ if $j < i \leq M$. We also use the notations $(i, j]$, $[i, j)$, and (i, j) for the intervals with one or both of the endpoints omitted.

DEFINITION 2.4. *Let R_i and B_j be nodes; we write $R_i \rightarrow B_j$ to denote a* directed *edge going from R_i forward (in tour order) to B_j. That is, we think of $R_i \rightarrow B_j$ jumping over the nodes $R_i, B_i, R_{i+1}, \ldots, R_j, B_j$. We say that $R_i \rightarrow B_j$ is a* candidate *(meaning, a candidate for a jumper), if*

$$c(R_i, B_j) + \sum_{\ell \in [i,j)} c'_\ell \;<\; \sum_{\ell \in [i,j]} c_\ell.$$

The intuitive meaning of $R_i \rightarrow B_j$ being a candidate is that it would be of lower cost to use the jumper $R_i \leftrightarrow B_j$, plus the greedy matching of adjacent nodes in (R_i, B_j), in place of just the greedy matching of adjacent nodes in $[R_i, B_j]$.

A similar definition is used to define what it means for an edge $B_i \rightarrow R_j$ to be a candidate; namely, $B_i \rightarrow R_j$ is a candidate if and only if

$$c(B_i, R_j) + \sum_{\ell \in (i,j)} c_i \;<\; \sum_{\ell \in [i,j)} c'_i.$$

Candidates always have endpoints of opposite colors and are directed. It is possible to have both $R_i \rightarrow B_j$ and $B_j \rightarrow R_i$ be (distinct) candidates or to have one or neither of them candidates.

It is an easy observation that if there are no candidates, then the greedy assignment(s) are minimum-cost matchings. To prove this, suppose σ is a minimum-cost matching which contains a jumper; by Lemma 2.3, σ may be picked to contain no crossing jumpers. Since there are no crossing jumpers, σ must contain a jumper $x_i \leftrightarrow x_j$ such that σ is greedy on (x_i, x_j) (namely, pick the jumper so as to minimize the tour-order distance from x_i to x_j). Let σ' be the matching which is the same as σ except greedy on $[x_i, x_j]$. Clearly σ' has one fewer jumper than σ, and since $x_i \rightarrow x_j$ is not a candidate, σ' has cost no greater than σ. Iterating this construction shows that at least one of the jumperless greedy matchings must be minimum-cost. To show they are both minimum-cost, let σ_0 and σ_1 be the greedy matchings which contain the edges $x_1 \leftrightarrow x_2$ and $x_1 \leftrightarrow x_N$, respectively. Then σ_0 cannot have cost lower than (respectively, higher than) the cost of σ_1 since otherwise, $x_2 \rightarrow x_1$ ($x_1 \rightarrow x_N$, respectively) would be a candidate.

DEFINITION 2.5. *A candidate $x_i \rightarrow x_j$ is a* minimal *candidate if and only if there is no other candidate $x_k \rightarrow x_\ell$ in its interior; that is to say, there is no candidate $x_k \rightarrow x_\ell$ with $[x_k, x_\ell]$ a proper subset of $[x_i, x_j]$.*

LEMMA 2.6. *Consider a balanced quasi-convex tour of alternating colors.*

[3]Note that, of the two greedy matchings for G, one is greedy on $[x_1, x_N]$ and the other is greedy on $[x_2, x_1]$.

(i) *Suppose $R_a \to B_b$ is a minimal candidate. Then every minimum-cost, crossing-free matching is greedy on the interval (R_a, B_b). That is to say, every minimum-cost, crossing-free matching contains the edges $B_{\ell-1} \leftrightarrow R_\ell$ for all $\ell \in (a, b]$.*

(ii) *Suppose $B_a \to R_b$ is a minimal candidate. Then every minimum-cost, crossing-free matching is greedy on the interval (B_a, R_b). That is to say, every minimum-cost, crossing-free matching contains the edges $R_\ell \leftrightarrow B_\ell$ for all $\ell \in (a, b)$.*

Note that Lemma 2.6 says only that the edges connecting adjacent nodes in the *interior* of the minimal candidate are in every minimum-cost matching; it does not say that the minimal candidate itself is a jumper in any minimum-cost matching. The proof of Lemma 2.6 is fairly involved and we postpone it until section 2.3. Lemma 2.6 also holds for line-like tours with alternating colors for candidates $x \to y$ with x, y in input order.

Lemma 2.6 suggests an algorithm for finding a minimum-cost matching. Namely, if there is a minimal candidate, greedily assign edges in its interior according to Lemma 2.6. This induces a matching problem on the remaining unassigned nodes, and it is clear that any minimum-cost matching on this smaller problem will lead to a minimum-cost matching for the original problem. Iterating this, one can continue removing nodes in the interiors of minimal candidates and reducing the problem size. Eventually a matching problem with no candidates will be reached; in this case, it suffices to greedily match the remaining nodes.

Unfortunately, this algorithm suggested by Lemma 2.6 is not linear time (yet); thus we need to refine Lemma 2.6 somewhat with the following definition.

DEFINITION 2.7. *We define*

$$\mathsf{Bnft}[R_a, B_b] = \left(\sum_{i \in [a,b]} c_i - \sum_{i \in [a,b)} c'_i \right) - c(R_a, B_b),$$

$$\mathsf{Bnft}[B_a, R_b] = \left(\sum_{i \in [a,b)} c'_i - \sum_{i \in (a,b)} c_i \right) - c(B_a, R_b),$$

and, for x and y the same color, $\mathsf{Bnft}[x, y] = -\infty$.

It is immediate that $\mathsf{Bnft}[x, y] > 0$ if and only if $x \to y$ is a candidate; in fact, $\mathsf{Bnft}[x, y]$ measures the benefit (i.e., the reduction in cost) of using $x \leftrightarrow y$ as a minimal jumper instead of the greedy matching on $[x, y]$.

The next lemma forms the basis for the correctness of the algorithm given in section 3 for the serial transitive closure problem. The general idea is that the algorithm will scan the nodes in tour order until at least one candidate is found and then, according to Lemma 2.8, the algorithm will choose an interval (x_ℓ, x_k) to greedily match. Once the interval (ℓ, k) has been greedily matched, the algorithm need only solve the induced matching problem on the remaining nodes.

LEMMA 2.8. *Let G be a balanced quasi-convex tour matching problem. Let $1 < k \le N$ and suppose $\mathsf{Bnft}[x_i, x_j] \le 0$ for all $1 \le i < j < k$. Further suppose that $m \stackrel{\text{def}}{=} \max\{\mathsf{Bnft}[x_i, x_k] : i < k\} > 0$ and let $\ell \stackrel{\text{def}}{=} \max\{i < k : \mathsf{Bnft}[x_i, x_k] = m\}$. Then every minimum-cost, crossing-free matching is greedy on (x_ℓ, x_k).*

Proof. The proof is, in essence, an iteration of Lemma 2.6. We argue by induction on k. Let G, k, m, and ℓ satisfy the hypothesis of the lemma. Let $s = \max\{i < k : \mathsf{Bnft}[x_i, x_k] > 0\}$, so $x_s \to x_k$ is a minimal candidate. By Lemma 2.6, any minimum-cost, crossing-free solution for G is greedy on the interval (x_s, x_k).

Hence, it will suffice to let G' be the matching problem obtained from G by discarding the nodes $x_{s+1}, \ldots, x_{k-1}$ and prove that any minimum-cost, crossing-free solution for G' is greedy on $(x_\ell, x_s]$. If $\ell = s$, there is nothing to prove, so we assume $\ell < s$. Note that x_k is now the $(s+1)$st node in the G' tour order. We use Bnft' to denote the Bnft function for G'.

We claim the following:

(i) If $1 \le i < j \le s$, $\mathsf{Bnft}'[x_i, x_j] = \mathsf{Bnft}[x_i, x_j]$.

(ii) If $1 \le i \le s$, $\mathsf{Bnft}'[x_i, x_k] = \mathsf{Bnft}[x_i, x_k] - \mathsf{Bnft}[x_s, x_k]$.

Claim (i) is immediate from the definition of Bnft. The intuitive meaning of (ii) is that the benefit of using the jumper $x_i \leftrightarrow x_k$ is reduced by the benefit already obtained from the jumper $x_s \leftrightarrow x_k$. We formally prove (ii) for the case that x_i and x_s are red and x_k is blue; the opposite colored case has a similar proof. Assume $x_i = R_a$, $x_s = R_b$, and $x_k = B_c$. Then

$$\mathsf{Bnft}'[R_a, B_c] = \sum_{\ell \in [a,b)} c_i + c(R_b, B_c) - \sum_{\ell \in [a,b)} c'_i - c(R_a, B_c),$$

$$\mathsf{Bnft}[R_a, B_c] = \sum_{\ell \in [a,c]} c_i - \sum_{\ell \in [a,c)} c'_i - c(R_a, B_c),$$

$$\mathsf{Bnft}[R_b, B_c] = \sum_{\ell \in [b,c]} c_i - \sum_{\ell \in [b,c)} c'_i - c(R_b, B_c).$$

From these three equations claim (ii) follows immediately.

Now let $m' = \max\{\mathsf{Bnft}'[x_i, x_k] : i < s\}$. By claim (ii), $m' = m - \mathsf{Bnft}[x_s, x_k]$; since $\ell < s$, $m' > 0$. Likewise, $\ell = \max\{i < s : \mathsf{Bnft}'[x_i, x_k] = m'\}$. Thus, by the induction hypothesis, any minimum-cost solution for G' is greedy on $(x_\ell, x_s]$ and Lemma 2.8 is proved. □

DEFINITION 2.9. *The Δ function is defined by*

$$\Delta[R_a, R_b] = \sum_{\ell \in [a,b)} c_\ell - \sum_{\ell \in [a,b)} c'_\ell,$$

$$\Delta[B_a, B_b] = \sum_{\ell \in [a,b)} c'_\ell - \sum_{\ell \in (a,b]} c_\ell.$$

LEMMA 2.10.

(i) $\mathsf{Bnft}[R_a, B_c] > \mathsf{Bnft}[R_b, B_c]$ *if and only if* $c(R_a, B_c) - c(R_b, B_c) < \Delta[R_a, R_b]$.

(ii) $\mathsf{Bnft}[B_a, R_c] > \mathsf{Bnft}[B_b, R_c]$ *if and only if* $c(B_a, R_c) - c(B_b, R_c) < \Delta[B_a, B_b]$.

Lemma 2.10 follows immediately from the definitions.

LEMMA 2.11. *Let u, v, x, y be in tour order with nodes u and v of one color and x and y of the other color. Then*

$$\mathsf{Bnft}[u, x] > \mathsf{Bnft}[v, x] \quad \Rightarrow \quad \mathsf{Bnft}[u, y] > \mathsf{Bnft}[v, y].$$

Proof. By Lemma 2.10, $\mathsf{Bnft}[u, x] > \mathsf{Bnft}[v, x]$ is equivalent to $c(u, x) - c(v, x) < \Delta[u, v]$, and $\mathsf{Bnft}[u, y] > \mathsf{Bnft}[v, y]$ is equivalent to $c(u, y) - c(v, y) < \Delta[u, v]$. Now, by quasi convexity, $c(u, x) - c(v, x) \ge c(u, y) - c(v, y)$, which suffices to prove the lemma. □

Let R_a and R_b be distinct red nodes. The previous two lemmas show that if there is any node B_c (with R_a, R_b, and B_c in tour order) such that $\mathsf{Bnft}[R_a, B_c]$ is greater than $\mathsf{Bnft}[R_b, B_c]$, then the first such B_c is the $\Delta[R_a, R_b]$-crossover point of

R_a and R_b. We shall denote this first B_c, if it exists, by $\chi[R_a, R_b]$; if it does not exist, then $\chi[R_a, R_b]$ is said to be undefined. Similarly, $\chi[B_a, B_b]$ is defined to the be the $\Delta[B_a, B_b]$-crossover point of B_a and B_b, and, if defined, is the first R_c where $\mathsf{Bnft}[B_a, R_c]$ is greater than $\mathsf{Bnft}[B_b, R_c]$.

We now assume that we have a procedure $\Omega(x, y, z)$, which, given nodes x, y, z in tour order, returns "True" if $\chi[x, y] \prec_y \chi[y, z]$ and returns "False" if $\chi[y, z] \prec_y \chi[x, y]$. (If neither condition holds, then $\Omega(x, y, z)$ may return an arbitrary truth value.) If the weak analyticity condition holds, then Ω is constant-time computable. Without this assumption, Ω is $O(\log N)$ time computable since Lemma 2.11 allows $\chi[-, -]$ to be computable by binary search.

The general idea of the algorithm given in section 3 below is that it will scan the nodes in tour order searching for candidates. Whenever a node is reached that is the head of a candidate, the algorithm will take the candidate specified in Lemma 2.8 (the one that was denoted $x_\ell \to x_k$) and greedily match the nodes in its interior. The greedily matched nodes are then dropped from consideration and the algorithm resumes its search for a candidate. Suppose the u and v are two nodes already scanned in this process that are being remembered as potential endpoints of candidates. Lemma 2.10 tells us that if a node x is found where $\mathsf{Bnft}[u, x] > \mathsf{Bnft}[v, x]$, then at all succeeding nodes y, $\mathsf{Bnft}[u, y] > \mathsf{Bnft}[v, y]$. By the criterion of Lemma 2.8, this means that after the node x is found, there is no further reason to consider candidates that begin at node v, since any candidate $v \to y$ would be subsumed by the better candidate $u \to y$.

To conclude this section we describe the algorithm in very general terms; in section 3 we give the precise specification of the algorithm. The algorithm scans nodes (starting with node x_1, say) and maintains three lists. The first list, $\mathcal{M}$, contains the nodes in tour order which have been examined so far. The second list, $\mathcal{L}^{-1}$, contains all the red nodes that need to be considered as potential endpoints of candidates (so $\mathcal{L}^{-1}$ is guaranteed to contain all the nodes satisfying the criterion of Lemma 2.8). The third list, $\mathcal{L}^1$, similarly contains all the blue nodes that need to be considered as potential endpoints of candidates. At any point during the scan, the lists will be of the form

$$\mathcal{M} = x_1, \ldots, x_{r-1},$$
$$\mathcal{L}^{-1} = R_{a_1}, \ldots, R_{a_p},$$
$$\mathcal{L}^{1} = B_{b_1}, \ldots, B_{b_q},$$

with $\mathcal{L}^{-1}$ and $\mathcal{L}^1$ subsequences of $\mathcal{M}$. The following five conditions will be maintained during execution:

(i) $x_1, \ldots, x_{r-1}$ are the nodes scanned but not matched and are in tour order, and there are no candidates $x_i \to x_j$ with $1 \le i < j < r$.

(ii) x_{r-1} precedes $\chi[R_{a_{p-1}}, R_{a_p}]$ in tour order.

(iii) For all $1 \le i \le p-2$, $\Omega(R_{a_i}, R_{a_{i+1}}, R_{a_{i+2}})$ is false.

(iv) For all $1 \le i \le q-2$, $\Omega(B_{b_i}, B_{b_{i+1}}, B_{b_{i+2}})$ is false.

(v) At any possible future node x_k following x_{r-1} such that x_k is the first point where a candidate is discovered; if the x_ℓ which satisfies Lemma 2.8 is among $x_1, \ldots, x_{r-1}$ then it is already on the list $\mathcal{L}^{-1}$ or $\mathcal{L}^1$ (depending on which color it is).

When scanning the next node x_r, the algorithm must do the following (we assume x_r is blue; similar actions are taken for red nodes):

(β) While $p \ge 2$ and $\mathsf{Bnft}[R_{a_{p-1}}, x_r] > \mathsf{Bnft}[R_{a_p}, x_r]$, pop R_{a_p} from $\mathcal{L}^{-1}$ and decrement p.

(γ) If $\mathsf{Bnft}[R_{a_p}, x_r] > 0$, greedily match nodes in the interval (R_{a_p}, x_r). The matched nodes are discarded from the lists $\mathcal{M}$, $\mathcal{L}^{-1}$, and $\mathcal{L}^1$ (the remaining nodes are to be implicitly renumbered at this point).

(δ) While $q \geq 2$ and $\Omega(B_{a_{q-1}}, B_{a_q}, x_r)$, pop B_{a_q} from $\mathcal{L}^1$ and decrement q. Then push x_r onto the end of $\mathcal{L}^1$ (and increment q).

Step (β) is justified by recalling that if x_r is past $\chi[R_{a_{p-1}}, R_{a_p}]$, then R_{a_p} may be removed from consideration as an endpoint of a candidate (by Lemma 2.8).

Step (δ) is justified as follows: suppose $R_i = \chi[B_{a_{q-1}}, B_{a_q}]$ equals or precedes $R_j = \chi[B_{a_q}, x_r]$ (using tour order, beginning at B_{a_q}). Then at any future candidate endpoint x_k, either x_k follows or equals R_i, in which case $\mathsf{Bnft}[B_{a_{q-1}}, x_k]$ is greater than $\mathsf{Bnft}[B_{a_q}, x_k]$, or x_k precedes R_j, in which case, $\mathsf{Bnft}[x_r, x_k]$ is greater than $\mathsf{Bnft}[B_{a_q}, x_k]$. Thus B_{a_q} will never be the starting endpoint of a candidate satisfying the criteria of Lemma 2.8, and we may drop it from consideration.

To justify step (γ) we must show that the candidate $R_{a_p} \to x_r$ satisfies the criteria from Lemma 2.8; in view of the correctness of the rest of the algorithm, for this it will suffice to show that $\mathsf{Bnft}[R_{a_i}, x_r] \leq \mathsf{Bnft}[R_{a_p}, x_r]$ for all $1 \leq i < p$. For this, note that step (β) and condition (iii) above ensure that x_r precedes $\chi[R_{a_i}, R_{a_{i+1}}]$ for all $1 \leq i < p$. This, in turn, implies $\mathsf{Bnft}[R_{a_i}, x_r] \leq \mathsf{Bnft}[R_{a_{i+1}}, x_r]$ for all i, which proves the desired inequality.

After the algorithm has scanned all the nodes once, it will have found and processed all candidates $x_i \to x_j$ where $i < j$. However, since the tour is circular, it is necessary to process candidates $x_i \to x_j$ with $i > j$. At the end of the first scan, the list $\mathcal{M}$ consists of all nodes $x_1, \ldots, x_n$ which have not been matched yet and $\mathcal{L}^{-1}$ and $\mathcal{L}^1$ contain nodes $R_{a_1}, \ldots, R_{a_p}$ and $B_{b_1}, \ldots, B_{b_q}$, as usual. During the second scan, the algorithm is searching for any candidates of the form $R_{a_i} \to B_j$ with $j < a_i$ or of the form $B_{a_i} \to R_j$ with $j \leq a_i$ (and only for such candidates). To process a node during the second scan, the algorithm pops x_1 off the left end of $\mathcal{M}$, implicitly renames x_1 to x_n and the rest of the nodes x_i to x_{i-1}, sets $r = n$, and does step (α) (still assuming x_r is blue):

(α) If x_r equals B_{b_1}, then pop B_{b_1} from the list $\mathcal{L}^1$ and implicitly renumber $\mathcal{L}^1$, decrementing q.

It then does steps (β)–(δ), except that in step (δ), the node x_r is not added to the end of $\mathcal{L}^1$. The reason for step (α) is that once a node B_{b_i} is encountered on the second scan, B_{b_i} is no longer a possible starting endpoint for a candidate. The reason for not adding x_r to the end of $\mathcal{L}^1$ in step (δ) is that it cannot be the starting endpoint of a candidate, because any such candidate would have already been found earlier.

The second scan will stop as soon as both $\mathcal{L}$ lists become empty. At this point no candidates remain and a greedy matching may be used for the remaining nodes in the $\mathcal{M}$ list.

The actual description of the algorithm with an efficient implementation is given in section 3, and it is there proved that the algorithm is linear time with the weak analyticity condition and $O(N \log N)$ time otherwise. Although we described steps (α)–(δ) only for blue x_r above, the algorithm in section 3 uses a toggle ψ to handle both colors with the same code. Finally, one more important feature of the algorithm is the way in which it computes the values of the Bnft function and of the $\Delta[x, y]$ function; it uses intermediate values $I[x]$ which are defined as follows.

DEFINITION 2.12. *The $I[x]$ function is defined by*

$$I[R_a] = \Delta[R_1, R_a],$$

$$I[B_a] = I[R_a] + c(R_a, B_a).$$

Note that $I[R_{a+1}] = I[B_a] - c(B_a, R_{a+1})$.

It is immediate from the definitions that, if x, y are tour order (starting from x_1), then

$$\begin{aligned}
\Delta[x,y] &= I[y] - I[x] && \text{for } x \text{ and } y \text{ red},\\
\Delta[x,y] &= I[x] - I[y] && \text{for } x \text{ and } y \text{ blue},\\
\mathsf{Bnft}[x,y] &= I[y] - I[x] - c(x,y) && \text{for } x \text{ red}, y \text{ blue},\\
\mathsf{Bnft}[x,y] &= I[x] - I[y] - c(x,y) && \text{for } x \text{ blue}, y \text{ red}.
\end{aligned}$$

These equalities permit the values of Δ and Bnft to be computed in constant time from the values of $I[-]$. Also, it is important to note that only the relative $I[-]$ values are needed; in other words, it is OK if the $I[-]$ values are shifted by a constant additive constant, since we always use the difference between two $I[-]$ values.

The $I[-]$ function is not only easy to compute but also provides an intuitive graphical means of understanding the above lemmas and algorithm description. For example, in Figure 2.1, $R_1 \to B_3$ is a (minimal) candidate whereas $R_1 \to B_1$ and $R_1 \to B_2$ are not candidates. In Figure 2.2(a), the node B_3 is the relative crossover, $\chi[R_1, R_2]$, of R_1 and R_2; on the other hand, in Figure 2.2(b), the relative crossover does not exist. Figure 2.3(a) shows an example where $\Omega(R_1, R_2, R_3)$ is true and Figure 2.3(b) shows an example where $\Omega(R_1, R_2, R_3)$ is false.

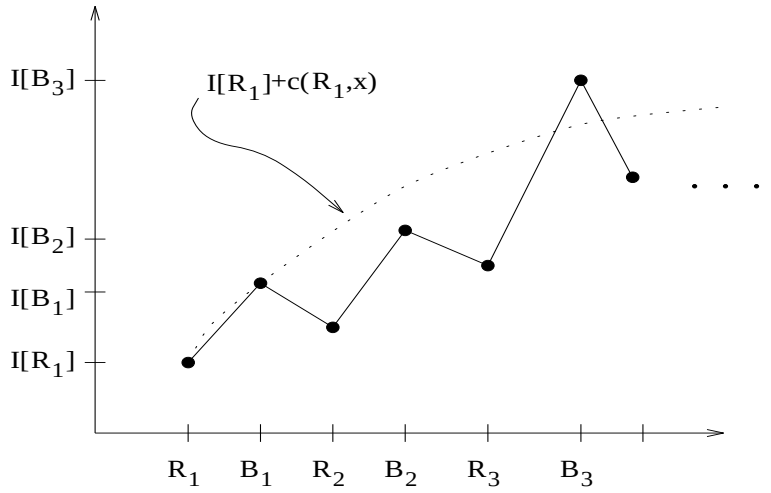


FIG. 2.1. *$R_1 \to B_3$ is a candidate as $I[R_1] + c(R_1, B_3) < I[B_3]$, which is equivalent to $\mathsf{Bnft}[R_1, B_3] > 0$.*

2.3. Proof of Lemma 2.6. By symmetry it will suffice to prove part (i). Since the lemma is trivial in case $a = b$, we assume $a \neq b$. Let σ be a crossing-free minimum-cost matching; we must prove that σ is greedy on (R_a, B_b). By the crossing freeness of σ and by the fact that $R_a \to B_b$ is a *minimal* candidate, σ does not contain any jumper with both endpoints in $[R_a, R_b]$, except possibly $R_a \leftrightarrow B_b$ itself. If $R_a \leftrightarrow B_b$ is in σ, then the same reasoning shows that σ is greedy on (R_a, B_b); so we suppose that $R_a \leftrightarrow B_b$ is not in σ. Since we are dealing (w.l.o.g.) with balanced tours, we may assume that $b = N$, by renumbering nodes if necessary.

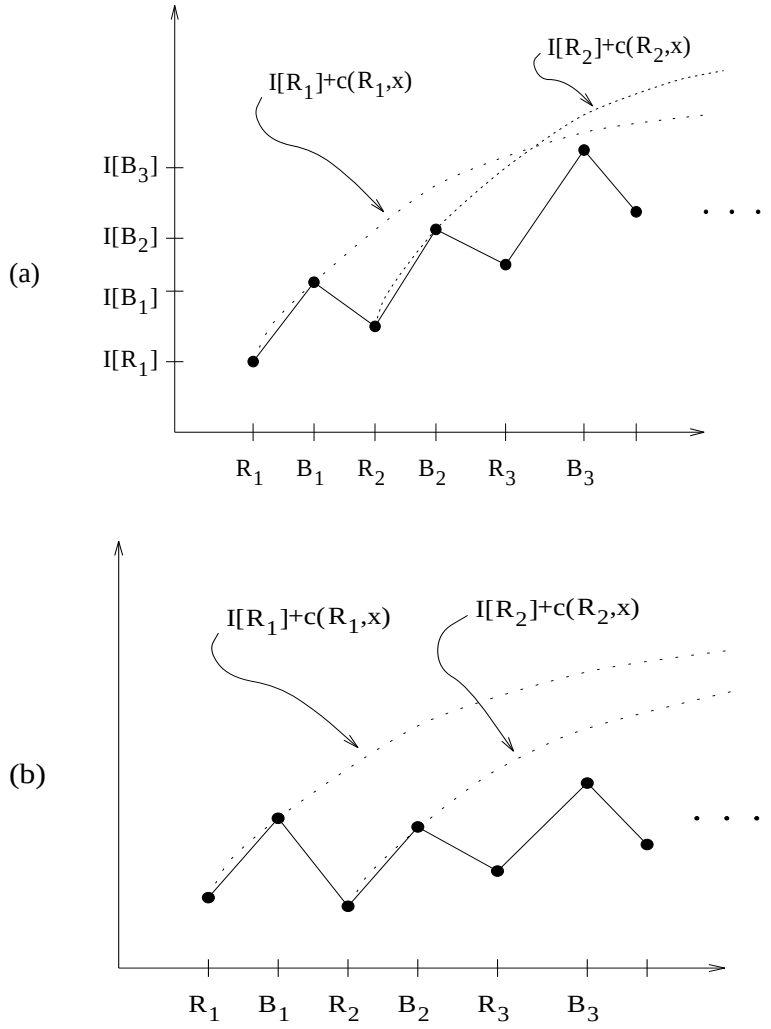


FIG. 2.2. *Illustrations of the relative crossover, $\chi[R_1.R_2]$, of R_1 and R_2. In* (a)*, B_3 is $\chi[R_1, R_2]$, since it is the first node x to satisfy $I[R_2]+c(R_2, x) > I[R_1]+c(R_1, x)$. In* (b)*, the relative crossover does not exist.*

Claim (i). $R_a \leftrightarrow B_a$ is not in σ.

Suppose, for a contradiction, that $R_a \leftrightarrow B_a$ is in σ. Let v be the least value such that $R_v \leftrightarrow B_q$ is in σ for some $q < a < v$. Note that such a v, $a < v \leq N$, must exist since there are no jumpers in $[R_a, B_N]$ and since σ is not greedy on $[R_a, B_N]$ (it cannot be greedy on $[R_a, B_N]$, since $R_a \rightarrow B_N$ is a candidate). By choice of v, σ is greedy on $[R_a, R_v)$. These edges in the matching σ are represented by edges drawn above the line in Figure 2.4(a). Since $R_a \rightarrow B_N$ is a minimal candidate, $\mathsf{Bnft}[R_a, B_N] > \mathsf{Bnft}[R_v, B_N]$, so Lemma 2.10 implies

$$\sum_{i\in[a,v)} c_i - \sum_{i\in[a,v)} c'_i > c(R_a, B_N) - c(R_v, B_N).$$

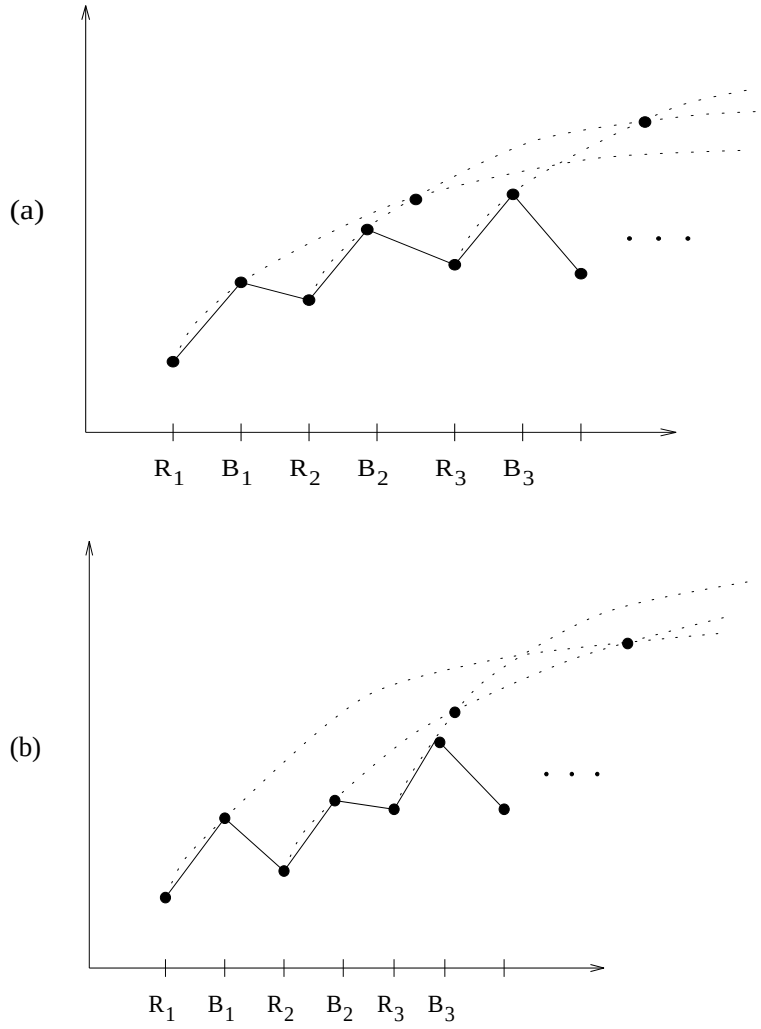


FIG. 2.3. $\Omega(R_1, R_2, R_3)$ *is true in* (a) *and false in* (b).

Since R_a, R_v, B_N, and B_q are in tour order, quasi convexity implies

$$c(R_a, B_N) - c(R_v, B_N) \geq c(R_a, B_q) - c(R_v, B_q).$$

Combining these inequalities yields

$$\sum_{i \in [a,v)} c_i + c(R_v, B_q) > \sum_{i \in [a,v)} c'_i + c(R_a, B_q). \tag{2.1}$$

Let σ' be the matching obtained from σ by replacing the jumper $B_q \leftrightarrow R_v$ and the greedy matching on $[R_a, R_v)$ with the edge $B_q \leftrightarrow R_a$ and the greedy matching on $(R_a, R_v]$. The new edges in σ' are drawn below the line in Figure 2.4(a). By (2.1), σ' has cost strictly less than the cost of σ, which is a contradiction.

Claim (ii). $R_N \leftrightarrow B_N$ is not in σ.

Claim (ii) is proved by an argument similar to Claim (i). Alternatively, reverse the colors and the tour order and Claim (ii) is a version of Claim (i).

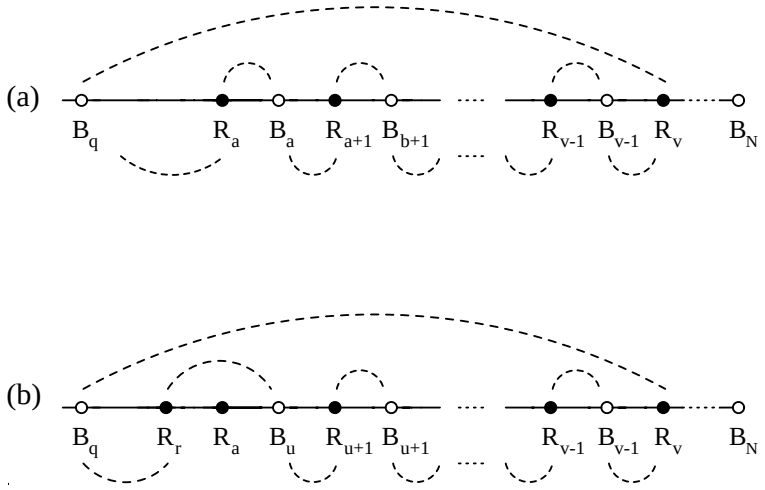


FIG. 2.4. *Illustrations of Claims* (i) *and* (iii) *from the proof of Lemma* 2.6. *The edges above the lines represent edges in the presumed minimum-cost matching σ; these are replaced by the edges below the line in the lower-cost matching σ'.*

Claim (iii). The matching σ is greedy on (R_a, B_b).

Suppose for a contradiction that σ is not greedy on (R_a, B_b). In view of Claims (i) and (ii) and since σ has no jumpers in $[R_a, B_N]$, this means that there exist u and v such that u is the least value, such that σ contains $B_u \leftrightarrow R_r$ with $r < a \leq u$, and v is the least value, such that σ contains $R_v \leftrightarrow B_q$ with $q < a \leq v$. Namely, let u be the least value $\geq a$ such that $B_u \leftrightarrow R_{u+1}$ is not in σ and v be the least value $> u$ such that $R_v \leftrightarrow B_v$ is not in σ. For these choices of u and v, it must be that $q < r < a \leq u \leq v \leq N$ and that σ is greedy on $[B_a, R_u]$ and $[R_{u+1}, B_{v-1}]$. These edges in the matching σ are represented by edges drawn above the line in Figure 2.4(b).

Since $R_a \rightarrow B_N$ is a candidate,

$$\sum_{i\in[a,N]} c_i > c(R_a, B_N) + \sum_{i\in[a,N)} c'_i.$$

And since it is minimal, neither $R_a \rightarrow B_u$ nor $R_v \rightarrow B_N$ are candidates; i.e.,

$$\sum_{i\in[a,u]} c_i \leq c(R_a, B_u) + \sum_{i\in[a,u)} c'_i,$$

$$\sum_{i\in[v,N]} c_i \leq c(R_v, B_N) + \sum_{i\in[v,N)} c'_i.$$

Combining these three inequalities gives

$$\sum_{i\in(u,v)} c_i + c(R_a, B_u) + c(R_v, B_N) > \sum_{i\in[u,v)} c'_i + c(R_a, B_N). \tag{2.2}$$

Since R_a, R_v, B_N, B_q and R_r, R_a, B_u, B_q are in tour order, quasi convexity implies the two inequalities

$$c(R_a, B_q) + c(R_v, B_N) \leq c(R_a, B_N) + c(R_v, B_q),$$

$$c(R_r, B_q) + c(R_a, B_u) \le c(R_r, B_u) + c(R_a, B_q)$$

which combine to yield

$$\begin{aligned} c(R_a, B_N) - c(R_a, B_u) - c(R_v, B_N) \\ \ge c(R_r, B_q) - c(R_r, B_u) - c(R_v, B_q). \end{aligned} \tag{2.3}$$

Using (2.2) and (2.3) gives the inequality

$$\sum_{i \in (u,v)} c_i + c(R_r, B_u) + c(R_v, B_q) > \sum_{i \in [u,v)} c'_i + c(R_r, B_q).$$

Let σ' be the matching obtained from σ by replacing the jumpers $R_r \leftrightarrow B_u$, $R_v \leftrightarrow B_q$, and the greedy matching on (B_u, R_v) with the edge $R_r \leftrightarrow B_q$ and the greedy matching on $[B_u, R_v]$. The new edges in σ' are drawn below the line in Figure 2.4(b). The last inequality above says that σ' has cost strictly less than the cost of σ, which is a contradiction. □

3. The algorithm. In this section, we give the actual algorithm for the main theorems. The correctness of the algorithm follows from the development in section 2.2. With Kanzelberger and Robinson, we have developed efficient implementations in ANSI-C of all the algorithms described below [5].[4]

3.1. Preliminaries. As mentioned above, the algorithm maintains three lists of nodes called deques (for "double ended queues," since we will have to access both ends of the lists). The three deques are the "main" deque $\mathcal{M}$ and two "left" deques $\mathcal{L}^1$ and $\mathcal{L}^{-1}$. The latter two are called "left deques" since they contain possible left endpoints for candidates. The deques will be updated by *push-right* operations which add a new node to the right end, by *pop-right* operations which pop the rightmost node off the deque, and by *pop-left* operations. However, *push-left* operations are never required. Deque operations can be efficiently implemented by using contiguous memory locations to store the deque elements and maintaining pointers to the left and right endpoints; each deque operation can then be performed in constant time. For our algorithm, it will suffice to reserve enough space for $2N$ deque elements (with no possibility that a deque will grow leftward since push-left's are not used).

Subscripts R, L, and $R-1$ are used to select the rightmost item, leftmost item, and the item preceding the rightmost, respectively. So $\mathcal{L}^{-1}_L$ refers to the leftmost element of $\mathcal{L}^{-1}$, $\mathcal{M}_{R-1}$ refers to the item just before the rightmost member of $\mathcal{M}$, etc. Each deque element is actually a pair, for example, $\mathcal{M}_R = (X, I)$; the first entry X of the pair is a node and the second entry I is a numerical value, namely $I = I[X]$ as defined in section 2.2. To simplify notation, we shall use the same notation for a deque element as for the node which is its first component. Thus, $\mathcal{M}_L$ also denotes the node which is its first component. We write $I[\mathcal{M}_L]$ to denote its second (numerical) component. Similar conventions apply to the $\mathcal{L}^{\pm 1}$ deques. To simplify our presentation of the algorithm, we deal with boundary effects by augmenting the definition of primitive operations as necessary. For example, accessing a nonexistent deque element will return an *undefined* indicator $\emptyset$ and, in general, functions of undefined operands are false or zero (in particular, the cost function $c(-,-)$ and the $I[-]$ functions return zero if they have $\emptyset$ as an argument).

[4]These C implementations are also available electronically from the authors or can currently be obtained by anonymous ftp from `math.ucsd.edu` or `ftp.nj.nec.com`

Function Input() returns the next vertex from an imagined input tape, which moves in the forward direction only and is assumed to hold a balanced alternating color tour. When the tape's end is reached, "*undefined*" is returned. Procedure Output() is used to write an individual matching to an imagined output tape. They are written as discovered but can easily be output in tour order (with only an extra $O(N)$ time computation).

To use the same code for red nodes and blue nodes, a variable ψ tracks vertex color by toggling between -1 and 1. Our convention is that $\psi = 1$ corresponds to blue and $\psi = -1$ to red.

3.2. Narrative description of the algorithm. Initialization consists of setting the three deques to be empty and setting the color toggle $\psi := -1$.

The algorithm first reads nodes from the input and pushes them onto the right end of the $\mathcal{M}$ deque, and then twice scans the nodes in tour order. During the two scans, nodes are popped from the left end of $\mathcal{M}$ and then pushed onto its right end.[5] In addition, while processing a node some nodes may be popped off the right end of $\mathcal{M}$ to be matched. It will always be the case that $\mathcal{M}$ contains a sequence of contiguous nodes in tour order and that the node currently being scanned immediately follows the (formerly) rightmost element of $\mathcal{M}$.

The variable ψ will be maintained as a color toggle, so that ψ is equal to -1 if the node currently being processed is red and to 1 if the current node is blue. The algorithm used for pushing an element onto the right end of $\mathcal{M}$ follows.

ALGORITHM 3.1. *This procedure pushes a vertex X onto the right of the $\mathcal{M}$ deque and computes the corresponding $I[X]$ value which is pushed along with X.*

procedure *Push_Main* (X)
    I:= $I[\mathcal{M}_R] + \psi \cdot c(\mathcal{M}_R, X)$
    push-right (X,I) *onto* $\mathcal{M}$
    return ()

Algorithm 3.1 merely computes the $I[-]$ value for a node X and pushes the node and its $I[-]$ value on the right end of M. To justify the computation of the value of $I[X]$, note that if X is blue, then $\psi = 1$ and $I[X]$ was defined to equal $I[\mathcal{M}_R] - c(\mathcal{M}_R, X)$; whereas, if X is red then $\phi = -1$ and $I[X]$ equals $I[\mathcal{M}_R] + c(\mathcal{M}_R, X)$. (Unless $\mathcal{M}$ is empty, in which case, $I[X] = 0$.)

Once the current node has been pushed onto the right end of $\mathcal{M}$, the following code implements step (β) from section 2.2:

while $c(\mathcal{L}_{R-1}^{\text{-}\psi}, \mathcal{M}_R) - c(\mathcal{L}_R^{\text{-}\psi}, \mathcal{M}_R) < \psi \cdot (I[\mathcal{L}_R^{\text{-}\psi}] - I[\mathcal{L}_{R-1}^{\text{-}\psi}])$
    pop-right $\mathcal{L}^{\text{-}\psi}$

To justify the correctness of the **while** condition, suppose that the currently scanned node is red, so $\phi = -1$. By Lemma 2.10, $\mathsf{Bnft}[\mathcal{L}_{R-1}^{\text{-}\psi}, \mathcal{M}_R] > \mathsf{Bnft}[\mathcal{L}_R^{\text{-}\psi}, \mathcal{M}_R]$ if and only if $c(\mathcal{L}_{R-1}^{\text{-}\psi}, \mathcal{M}_R) - c(\mathcal{L}_R^{\text{-}\psi}, \mathcal{M}_R) < \Delta[\mathcal{L}_{R-1}^{\text{-}\psi}, \mathcal{L}_R^{\text{-}\psi}]$. Furthermore, $\Delta[\mathcal{L}_{R-1}^{\text{-}\psi}, \mathcal{L}_R^{\text{-}\psi}]$ is equal to $\psi \cdot (I[\mathcal{L}_R^{\text{-}\psi}] - I[\mathcal{L}_{R-1}^{\text{-}\psi}])$ since $\mathcal{L}^{\text{-}\psi}$ contains blue nodes and $\psi = -1$ (by the equalities at the end of section 2.2). In this case, $\mathcal{M}_R$ is past the crossover point of $\mathcal{L}_{R-1}^{\text{-}\psi}$ and $\mathcal{L}_R^{\text{-}\psi}$, so $\mathcal{L}_R^{\text{-}\psi}$ may be discarded from consideration as a left endpoint of a candidate. A similar calculation justifies the case when the current node is blue.

[5] For line-like tours, only the first scan is needed; however, we treat only the more general (circular) case.

To implement step (γ), the following code is used:

if $c(\mathcal{M}_R, \mathcal{L}_R^{\neg\psi}) < \psi \cdot (I[\mathcal{M}_R] - I[\mathcal{L}_R^{\neg\psi}])$
  X := *pop-right* $\mathcal{M}$
  while $\mathcal{M}_R \neq \mathcal{L}_R^{\neg\psi}$
    Match_Pair()
  Push_Main(X)

where Match_Pair is defined below. The above **if** statement checks whether $\mathcal{L}_R^{\neg\psi} \rightarrow \mathcal{M}_R$ is a candidate; if so, the algorithm greedily assigns edges to nodes in the interior of the candidate (where "greedily" means with respect to the nodes that have not already been assigned). Before the greedy assignment is started, the rightmost entry is popped from $\mathcal{M}$ and is saved as X to pushed back on the right end afterwards. There are two reasons for this: first, this gets the current node X out of the way of Match_Pair's operation, and second, and more importantly, when X is pushed back onto $\mathcal{M}$, the $I[-]$ value for the current node is recomputed so as to be correct for the reduced matching problem in which the greedily matched nodes are no longer present. Match_Pair is the following procedure:

procedure *Match_Pair*()
  Output ("$\mathcal{M}_{R-1} \leftrightarrow \mathcal{M}_R$")
  pop-right $\mathcal{M}$
  if $\mathcal{M}_R = \mathcal{L}_R^{\psi}$
    pop-right $\mathcal{L}^{\psi}$
  pop-right $\mathcal{M}$
  return()

The procedure Match_Pair assigns a jumper $\mathcal{M}_{R-1} \leftrightarrow \mathcal{M}_R$ and discards a matched node from the deque $\mathcal{L}^{\psi}$ if it appears there. Because of the **while** condition controlling calls to Match_Pair, it is not possible for a matched node to occur in $\mathcal{L}^{\neg\psi}$, so we do not check for this condition.

To implement step (δ), the following code is used:

while $\Omega(\mathcal{L}_{R-1}^{\psi}, \mathcal{L}_R^{\psi}, \mathcal{M}_R) =$ "Yes"
  pop-right $\mathcal{L}^{\psi}$
push-right $\mathcal{M}_R$ onto $\mathcal{L}^{\psi}$ (without popping $\mathcal{M}_R$)

That completes the description of how nodes are processed during the first scan. As mentioned earlier, the last instruction (the *push-right*) is omitted from step (δ) during the second scan. Other than this, the processing for steps (β)–(δ) is identical in the two scans.

One potentially confusing aspect of the second scan is that the $I[-]$ values are no longer actually the correct $I[-]$ values; for example, it is no longer the case that $I[\mathcal{M}_L]$ is necessarily equal to zero. Strictly speaking, the $I[-]$ values all shift by an additive constant when an entry is popped from the left end of $\mathcal{M}$; however, it is not necessary to implement this shift, since the algorithm only uses differences between $I[-]$ values. The end result is that nothing special needs to be done to the I values when we pop-left $\mathcal{M}$.

After both scans are completed, any remaining nodes may be greedily matched. As discussed above, there are two possible greedy matchings and both have the same (optimal) cost. Thus either one may be used; the algorithm below just calls

Match_Pair repeatedly to assign one of these greedy matchings.

ALGORITHM 3.2. *This is the matching algorithm for balanced quasi-convex tours. All variables are global.*

"Initialization"
$\mathcal{M}, \mathcal{L}^{-1}, \mathcal{L}^{1} := \emptyset$
ψ := *-1*
"Read Input into the $\mathcal{M}$ deque"
while $[X :=$ *Input*$()] \neq \emptyset$
  Push_Main (X)
  $\psi := -\psi$
"The First Scan"
while $\mathcal{L}^{\psi}$ *is empty or* $\mathcal{M}_L \neq \mathcal{L}^{\psi}_L$
  $X :=$ *pop-left* $\mathcal{M}$
  Process_Node()
  push-right $\mathcal{M}_R$ *onto* $\mathcal{L}^{\psi}$
  $\psi := -\psi$
"The Second Scan"
while $\mathcal{L}^{-1}$ *and* $\mathcal{L}^{1}$ *are not both empty*
  $X :=$ *pop-left* $\mathcal{M}$
  if $X = \mathcal{L}^{\psi}_L$
    pop-left $\mathcal{L}^{\psi}$
  Process_Node()
  $\psi := -\psi$
"Windup Processing"
while $\mathcal{M}$ *is not empty*
  Match_Pair()
Exit.
procedure *Process_Node*()
  Push_Main(X)
  while $c(\mathcal{L}^{\bar{\psi}}_{R-1}, \mathcal{M}_R) - c(\mathcal{L}^{\bar{\psi}}_{R}, \mathcal{M}_R) < (I[\mathcal{L}^{\bar{\psi}}_{R}] - I[\mathcal{L}^{\bar{\psi}}_{R-1}]) \cdot \psi$
    pop-right $\mathcal{L}^{\bar{\psi}}$
  if $c(\mathcal{M}_R, \mathcal{L}^{\bar{\psi}}_{R}) < \psi \cdot (I[\mathcal{M}_R] - I[\mathcal{L}^{\bar{\psi}}_{R}])$
    $X :=$ *pop-right* $\mathcal{M}$
    while $\mathcal{M}_R \neq \mathcal{L}^{\bar{\psi}}_{R}$
      Match_Pair()
    Push_Main(X)
  while $\Omega(\mathcal{L}^{\psi}_{R-1}, \mathcal{L}^{\psi}_{R}, \mathcal{M}_R)$
    pop-right $\mathcal{L}^{\psi}$
  return

The complete matching algorithm is shown as Algorithm 3.2. When interpreting Algorithm 3.2, it is necessary to recall our convention that any predicate of undefined arguments is to be false. This situation can occur in the four **while** and **if** conditions of Process_Node. If $\mathcal{L}^{\bar{\psi}}$ is empty, then $\mathcal{L}^{\bar{\psi}}_{R}$ is undefined and the two **while** conditions and the first **if** condition are to be false. Similarly, if $\mathcal{L}^{\bar{\psi}}$ has < 2 elements, then the first **while** condition is to be false; and if $\mathcal{L}^{\psi}$ has < 2 elements, then the final **while** condition is to be false.

The runtime of Algorithm 3.2 is either $O(N)$ or $O(N \log N)$ depending on whether the weak analyticity condition holds. To see this, note that the initialization and the windup processing both take $O(N)$ time. The loops for each of the two scans are executed $\leq N$ times. Except for the **while** loops, each call to Process_Node takes constant time. The second **while** loop (which calls Match_Pair) is executed more than once only when edges are being output. If the first or third **while** loop is executed more than once, then vertices are being popped from the L stacks. Since $\lfloor \frac{1}{2}N \rfloor$ edges are output and since $O(N)$ vertices are pushed onto the L stacks, each of these **while** loops are executed only $O(N)$ times during the *entire* execution of the algorithm. An iteration of the first or second **while** loop takes constant time, while an iteration of the third **while** loop takes either constant time or $O(\log N)$ time, depending on whether the weak analyticity property holds.

When the weak analyticity condition holds, the Ω predicate typically operates in constant time by computing two theoretical relative crossovers and comparing their positions. This happens, for example, when the tour consists of points lying on a circle, with the cost function equal to Euclidean distance; section 3.3 outlines a constant-time algorithm for this example. Without the weak analyticity condition, the Ω-predicate runs in logarithmic time, by using a binary search of the $\mathcal{M}$ deque. This general (not weakly analytic) case is handled by the generic Ω algorithm discussed in section 3.3.

There are a couple of improvements that can be made to the algorithm which will increase execution speed by a constant factor. First, the calls to Match_Pair made during the "Windup Processing" do not need to check if $\mathcal{M}_R = \mathcal{L}_R^\psi$, since $\mathcal{L}^\psi$ is empty at this time. Second, if computing the cost function $c(-,-)$ is more costly than simple addition, then it is possible for Push_Main() to use an alternative method during the two scans to compute the cost $c(\mathcal{M}_R, X)$ for nodes X which have just been popped from the left of $\mathcal{M}$ (except for the first one popped from the left in the first scan). Namely, the algorithm can save the old $I[X]$ value for the node X as it is left-popped off the deque $\mathcal{M}$. Then the cost function can be computed by computing the difference between the $I[-]$ value of X and the $I[-]$ of the previous node left-popped from $\mathcal{M}$. This second improvement applies only to the first Push_Main call in Process_Node.

3.3. Algorithms for Ω. During the first scan, the procedure Ω is called (repeatedly) by Process_Node to determine whether $\mathcal{L}_R^\psi$ should be popped before the current node $\mathcal{M}_R$ is pushed onto the right end of the $\mathcal{L}^\psi$ deque. During the second scan, $\mathcal{M}_R$ is never pushed onto the $\mathcal{L}^\psi$ deque; however, using the procedure Ω can allow the $\mathcal{L}^\psi$ deque to be more quickly emptied, thus speeding up the algorithm's execution. (However, the use of the Ω could be omitted during the second scan without affecting the correctness of the algorithm.)

When Ω is called, $\mathcal{L}_{R-1}^\psi$, $\mathcal{L}_R^\psi$, and $\mathcal{M}_R$ are distinct nodes, in tour order, and of the same color. Let $\delta = (I[\mathcal{L}_{R-1}^\psi] - I[\mathcal{L}_R^\psi]) \cdot \psi$ and $\epsilon = (I[\mathcal{L}_R^\psi] - I[\mathcal{M}_R]) \cdot \psi$. Let Y denote the δ-crossover point of $\mathcal{L}_{R-1}^\psi$ and $\mathcal{L}_R^\psi$, and let Z denote the ϵ-crossover point of $\mathcal{L}_R^\psi$ and $\mathcal{M}_R$ (note Y and/or Z may not exist). By definition, Y and Z are both opposite in color from the other three nodes. The procedure Ω must return *True* if Y exists and Z does not, must return *False* if Y does not exist, and, if both exist, must return *True* if $\mathcal{L}_R^\psi, Y, Z$ are in tour order, must return *True* if $\mathcal{L}_R^\psi, Z, Y$ are in tour order, and may return either value if $Y = Z$.

In this section, we discuss two algorithms for Ω. We first discuss a "generic" algorithm that works for any cost function, regardless of whether the weak analyticity condition holds. This generic Ω procedure is shown as Algorithm 3.3 below. The

generic Ω executes a binary search for a node which is at or past one of the crossover points but is not at or past the other. Obviously, if the crossover Y exists, then it exists in the range $(\mathcal{L}_R^\psi, \mathcal{L}_{R-1}^\psi)$, and if the crossover Z exists, it is in the range $(\mathcal{M}_R, \mathcal{L}_R^\psi)$. Furthermore, Y cannot exist in the range $(\mathcal{L}_R^\psi, \mathcal{M}_R)$, since otherwise it would have been popped when Y was reached (by the first **while** loop in an earlier call to Match_Pair). Hence, the binary search may be confined to the range $(\mathcal{M}_R, \mathcal{L}_{R-1}^\psi)$, provided that *True* is returned in the event that the binary search is unsuccessful.

In Algorithm 3.3, a new notation $\mathcal{M}_k$ is used. This presumes that the $\mathcal{M}$ deque is implemented as an array; the elements of $\mathcal{M}$ fill a contiguous block of the array elements. When we write $\mathcal{M}_k$, we mean the kth entry of the array. The L deques can contain pointers to $\mathcal{M}$ deque entries; in fact, in our preferred implementation, the L deque entries contain only an index for an $\mathcal{M}$ deque entry. Thus the value h can be found in constant time for Algorithm 3.3.

The generic Ω algorithm shown in Algorithm 3.3 takes $O(\log N)$ time since it uses a binary search.

ALGORITHM 3.3. *This is the generic Ω algorithm which works with any tour, regardless of weak analyticity. Past_Xover_A and Past_Xover_B are boolean-valued variables.*

procedure $\Omega(\mathcal{L}_{R-1}^\psi, \mathcal{L}_R^\psi, \mathcal{M}_R)$

"Nodes $\mathcal{L}_{R-1}^\psi$, $\mathcal{L}_R^\psi$ and $\mathcal{M}_R$ are the same color."

Let h be the index so that $\mathcal{M}_{h-1}$ is $\mathcal{L}_{R-1}^\psi$.
Let ℓ be the index so that $\mathcal{M}_\ell$ is $\mathcal{M}_R$.

$\delta := (I[\mathcal{L}_{R-1}^\psi] - I[\mathcal{L}_R^\psi]) \cdot \psi$
$\epsilon := (I[\mathcal{L}_R^\psi] - I[\mathcal{M}_R]) \cdot \psi$

"Do binary search of opposite color nodes from $\mathcal{M}_\ell$ to $\mathcal{M}_{h-2}$"

while $h > \ell + 1$
  $k := \ell + 2\lfloor (h-\ell)/4 \rfloor$
  Past_Xover_A:= $(c(\mathcal{L}_{R-1}^\psi, \mathcal{M}_k) - c(\mathcal{L}_R^\psi, \mathcal{M}_k)) < \delta$
  Past_Xover_B:= $(c(\mathcal{L}_R^\psi, \mathcal{M}_k) - c(\mathcal{M}_R, \mathcal{M}_k)) < \epsilon$
  if *Past_Xover_A*
    if *Past_Xover_B*
      $h := k$
    else
      return(TRUE)
  else
    if *Past_Xover_B*
      return(FALSE)
    else
      $\ell := k + 2$
return (TRUE)

Next we describe an example of a linear time algorithm for Ω where the weak analyticity condition holds. For this example, we assume that the nodes of the quasiconvex tour lie on the unit circle in the xy plane, the cost function is equal to straight-line Euclidean distance, and the tour proceeds in counterclockwise order around the circle. The Ω algorithm either is given, or computes, the xy coordinates of the three nodes $\mathcal{L}_{R-1}^\psi$, $\mathcal{L}_R^\psi$, and $\mathcal{M}_R$. It then uses a routine Circle_Crossover to find the **theoretical**

δ-crossover Y of $\mathcal{L}_{R-1}^{\psi}$ and $\mathcal{L}_R^{\psi}$ and the **theoretical** ϵ-crossover of $\mathcal{L}_R^{\psi}$ and $\mathcal{M}_R$. If the theoretical crossover Y exists, it will be in the interval $[\mathcal{L}_R^{\psi}, \mathcal{L}_{R-1}^{\psi}]$, and if Z exists, Z will be in the interval $[\mathcal{M}_R, \mathcal{L}_R^{\psi}]$. When Y and Z both exist, the Ω procedure returns *True* if $\mathcal{L}_R^{\psi}, Y, Z$ are in tour order or any two of these nodes are equal; otherwise the procedure returns *False*.

ALGORITHM 3.4. *This is the algorithm which computes the δ-theoretical crossover point for two nodes lying on the unit circle with cost function equal to Euclidean distance. The inputs are δ and two points (x_1, y_1) and (x_2, y_2) lying on the unit circle. The procedure returns TRUE or FALSE to indicate whether the crossover point exists; if it does exist, it sets (x_3, y_3) equal to the crossover point. There is a possibility that roundoff errors will lead to spurious "FALSE" answers, so the values of (x_3, y_3) are set even when FALSE is returned.*

```
procedure Circle_Crossover(x1,y1,x2,y2,δ)
    a := δ/2
    hip := (x1 · x2 + y1 · y2)/2
    if hip > 0.5                    "two checks to avoid roundoff errors"
        hip := 0.5
    else if hip < −0.5
        hip := −0.5
    csqr := .5 − hip
    c := √csqr
    d := √(.5 + hip)
    if −x2 · y1 + x1 · y2 < 0
        d := −d
    asqr := a · a
    if asqr > csqr
        if a < 0
            x3 := x1
            y3 := y1
        else
            x3 := x2
            y3 := y2
        return(FALSE)
    u := −(1 + d)(1 − (asqr/csqr))
    if asqr = csqr
        v := a
    else
        v := a√(1 + (u²)/(csqr − asqr))
    if hip > 0
        α := (x1 + x2)/(2d)
        β := (y1 + y2)/(2d)
    else
        α := (y2 − y1)/(2c)
        β := (x1 − x2)/(2c)
    x3 := (u + d)α − vβ
    y3 := vα + (u + d)β
    return (TRUE)
```

Of course, the crucial implementation difficulty for the procedure Ω is the algorithm for Circle_Crossover; this is shown as Algorithm 3.4. Circle_Crossover takes two points (x_1, y_1) and (x_2, y_2) lying on the unit circle in the xy-plane and a real

value δ. The theoretical δ-crossover point of (x_1, y_1) and (x_2, y_2) is found as an intersection point of the unit circle and the hyperbola consisting of those points which have distance from (x_1, y_1) equal to δ plus their distance from (x_2, y_2) (namely, the intersection which is not between (x_1, y_1) and (x_2, y_2) in tour order). Letting the "half inner product" hip equal $(x_1x_2+y_1y_2)/2$, the distance between the two points is equal to $2c$, where $c = \sqrt{\frac{1}{2} - hip}$. And, the midpoint of the line segment between the two points is distance $\sqrt{\frac{1}{2} + hip}$ from the origin. To conveniently express the equation for the hyperbola, we set up uv-axes as a rigid translation of the xy-axes, positioned so that the points (x_1, y_1) and (x_2, y_2) have uv-coordinates $(0, -c)$ and $(0, c)$, respectively. This makes the origin have uv-coordinates $(d, 0)$, where $d = \pm\sqrt{\frac{1}{2} + hip}$ with the sign being + if and only if the angle from (x_1, y_1) to (x_2, y_2) is ≤ 180 deg. In the uv-plane, the hyperbola has equation

$$\frac{v^2}{a^2} - \frac{u^2}{c^2 - a^2} = 1$$

where $a = \delta/2$, and the unit circle has equation

$$(u + d)^2 + v^2 = 1.$$

Eliminating v^2 from these equations and solving for u, and then for v, shows that the desired intersection point of the circle and the hyperbola has uv-coordinates

$$u = (1 - d)\left(1 - \frac{a^2}{c^2}\right),$$
$$v = a\sqrt{1 + \frac{u^2}{c^2 - a^2}}.$$

Given the uv-coordinates, it is an easy matter to find values α, β which allow the corresponding xy-coordinates to be computed. Algorithm 3.4 show two equivalent calculations of α, β; the algorithm chooses the one which avoids division by zero or division by a number close to zero. Algorithm 3.4, as shown, also checks for some error conditions that can arise from roundoff errors. In particular, it makes sure that $|hip| \leq \frac{1}{2}$, and that $a^2 < c^2$. Amazingly enough, we found, during extensive testing with randomly generated tours of points on the unit circle, that roundoff error occasionally caused these conditions to be violated, even for points on the unit circle and for $|a| < |c|$.

3.4. An ANSI-C implementation. An efficient and highly portable ANSI-C implementation of our algorithms is described in [5], which includes complete source code, test programs for several interesting cases, benchmark results, and software to produce postscript graphical representations of the matchings found. To help ensure the correctness of our implementation, a straightforward $O(n^3)$ dynamic programming solution was also implemented, and the results compared for 4,000,000 pseudo-randomly drawn problems. Figure 1.1 shows an example of a matching produced by our software.

Benchmark results for a variety of RISC processors produced nearly identical results when normalized by clock rate. So timing results in [5] are given in units of RISC cycles. Graphs of up to 20,000 nodes are included in this study.

Recall that $O(\log N)$ time is a worst case bound for generic Ω. One interesting experimental result is that over the range of graph sizes considered, for the specific settings implemented in the test programs, and given the uniform pseudorandom manner in which problem instances were generated, the generic Ω implementation exhibits very nearly linear time performance. In other words, the experimentally observed runtime of Ω was nearly constant. We suspect that this is primarily a consequence of the uniform random distribution from which problems were drawn, and that it should be possible to demonstrate expected time results better than $O(\log N)$ for more structured settings.

The benchmarks included one line-like and two circular settings. Solving pseudorandomly drawn matching problems of size n required on average between 2,000 and 16,000 RISC cycles per node depending on the setting and on whether a constant-time or generic Ω was employed. It is interesting to note that, in all cases, the constant-time Ω performed better by a factor ranging from roughly 1.5 to slightly over 3. Thus, for some problems, the linear time result of this paper may be of practical interest.

Despite our focus on efficiency, further code improvements and cost function evaluation by table lookup may contribute to significant performance improvement.

4. Nonbipartite, quasi-convex tours. In this section we show how the earlier algorithms can be applied to nonbipartite, quasi-convex tours. The principal observation is that nonbipartite tours may be made bipartite by the simple construction of making the nodes alternate in color. This is already observed by Marcotte and Suri [20] in a more restrictive setting; we repeat the construction here for the sake of completeness.

First, it is apparent that the proof of Lemma 2.3 still works in the nonbipartite case, and thus any nonbipartite, quasi-convex tour has a minimum-cost matching in which no jumpers cross. This fact implies the following two lemmas.

LEMMA 4.1. *Let $x_1, \ldots, x_N$ be a nonbipartite, quasi-convex tour with N even. Then there exists a minimum-cost matching such that every edge in the tour is of the form $x_i \leftrightarrow x_j$ with i even and j odd.*

Proof. It will suffice to show that any crossing-free matching has this property. Suppose $x_i \leftrightarrow x_j$ is a jumper in a crossing-free matching, with $i < j$. Since N is even, the matching is complete in that every node is matched. The crossing-free property thus implies that the nodes in (x_i, x_j) are matched with each other, so there are an even number of such nodes, i.e., one of i and j is even and the other is odd. □

LEMMA 4.2. *Let $x_1, \ldots, x_N$ be a nonbipartite line-like quasi-convex tour. Then there exists a minimum-cost matching such that every edge in the tour is of the form $x_i \leftrightarrow x_j$ with i even and j odd.*

Proof. If N is even then this lemma is just a special case of the former lemma. If N is odd, then add an additional node x_{N+1} to the end of the tour, with $c(x_i, x_{N+1}) = 0$ for all i. The resulting tour is again quasi-convex and of even length; so the lemma again follows immediately from the former lemma. □

When Lemmas 4.1 and 4.2 apply, we may color the even nodes red and the odd nodes blue and reduce the nonbipartite matching problem to a bipartite matching problem. As an immediate corollary, we have that the two main theorems also apply in the nonbipartite setting; namely, for nonbipartite, quasi-convex tours of even length and for nonbipartite, line-like, quasi-convex tours, the matching problem can always be solved in $O(N \log N)$ time and it can be solved in $O(N)$ time if the weak analyticity condition holds.

We do not know whether similar algorithms exist for the case of general (i.e.,

nonline-like) quasi-convex tours of odd length. Similarly, we do not know any linear or near-linear time algorithms for bipartite, quasi-convex tours which are neither balanced nor line-like.

We conclude this section by mentioning a tantalizing connection between our work and the work of Yao [25]. Yao gave a quadratic runtime algorithm for solving the dynamic programming problem

$$d(i,j) \;=\; c(i,j) + \min\{d(i,k-1) + d(k,j) : i < k \le j\}$$

for line-like quasi-convex tours with cost function c (improving on the obvious cubic-time algorithm). Our nonbipartite matching problem can be stated as a similar dynamic programming problem; namely, the minimum-cost, $MC(i,j)$, of a complete matching on the nodes in $[x_i, x_j]$ can be recursively defined to equal

$$\min\{c(i,k) + MC(i+1,k-1) + MC(k+1,j) : i < k \le j\}.$$

(A similar dynamic programming algorithm can be given for the bipartite matching problem.) The obvious naive algorithm for computing $MC(-,-)$ is cubic-time; however, our main results give (near)-linear time algorithms for line-like quasi-convex tours. This raises the possibility that the dynamic programming problem considered by Yao may also have a near-linear time solution.

5. Applications to string matching. As a final topic we briefly discuss the application of our matching results to string comparison. A full treatment is beyond the scope of this paper, but additional details and related algorithms may be found in [6]. Given two symbol strings $v = a_1 a_2 \cdots a_n$ and $w = b_1 b_2 \cdots b_n$, our goal is to measure a particular notion of *distance* between them. Intuitively, distance acts as a measure of similarity; i.e., strings that are highly similar (highly dissimilar) are to have a small (large) distance between them. The purpose of such formulations is usually to approximate human similarity judgments within a pattern classification or information retrieval system.

Suppose $f(x)$ is a monotonely increasing, concave-down function with $f(0) = 0$. Let symbols $a_1, \ldots, a_n$ in v be a graph's red nodes and $b_1, \ldots, b_n$ in w be its blue nodes, and consider bipartite matchings of these $2n$ symbols. In the simplest formulation, we define the cost of an edge $a_i \leftrightarrow b_j$ as $f(|j-i|)$ if a_i and b_j are the same symbol and as $f(n)$ if a_i and b_j are distinct symbols. The cost of matching unequal characters can also be set to be any other fixed value instead of $f(n)$. Our *distance*, $\sigma(v,w)$, between strings v and w is then the minimum cost of any such bipartite matching.

As an example, consider the two strings "delve" and "level" and let $f(x) = \sqrt{x}$. Then the distance between these two strings is $\sqrt{5} + \sqrt{0} + \sqrt{2} + \sqrt{1} + \sqrt{1} \approx 5.65$.

As we have set up our problem above, the computation of $\sigma(v,w)$ is not directly an instance of the quasi-convex matching problem. However we can compute the σ function by considering each alphabet symbol α separately, and solving the quasi-convex matching problem σ_α which results from restricting attention to occurrences of a single alphabet symbol at a time. To make this clear, we introduce a special symbol "-" which indicates the absence of an alphabet symbol. The value of $\sigma($"delve," "level"$)$ can be expressed as the sum

$$\begin{aligned}&\sigma_d(\text{“}d\text{----”},\text{“-----”}) + \sigma_e(\text{“-}e\text{--}e\text{”},\text{“-}e\text{-}e\text{-”})\\ &\quad +\sigma_l(\text{“--}l\text{--”},\text{“}l\text{---}l\text{”}) + \sigma_v(\text{“---}v\text{-”},\text{“--}v\text{--”}).\end{aligned}$$

To make the summed σ_α terms equal σ as originally defined, each σ_α is defined to be the subproblem's minimum matching cost plus $f(n)/2$ times the number of unmatched symbols.

We will loosely refer to distance functions that result from this kind of formulation as σ-distances. Assuming that $f(x)$ satisfies the weak analyticity condition, it is not too difficult to show that it is possible to compute $\sigma(v, w)$ in linear time. If the weak analyticity condition does not hold, then our results give an $O(n \log n)$ time algorithm.

A novel feature of our σ-distances is that distinct alphabet symbols are treated independently. This is in contrast to most prior work which has used "least edit distance" for string comparison (see [21] for a survey). As an illustration of the difference between our distance measure and the "edit distance" approach, consider comparing the word "abcde" with its mirror image "edcba." Our approach recognizes some similarity between these two forms, while the most standard "edit distance" approach sees only that the two strings have "c" in common—in essence substituting the first two and last two symbols of the string without noticing the additional occurrences of the same symbols at the other end of the other string.

A special form of our σ-distance measure in which $f(x) = x$, and the optimal matching is only approximated, was introduced earlier by the authors and shown to have a simple linear time algorithm [26, 27]. Its relationship to σ-distances is described in [6]. This earlier algorithm has been successfully used in commercial applications, especially for spelling correction in word processing software, typewriters, and handheld dictionary devices (we estimate that over 15,000,000 such software/hardware units have been sold by Proximity Technology, Franklin Electronic Publishers, and their licensees). Other less prominent commercial applications include database field search (e.g., looking up a name or address), and the analysis of multifield records such as mailing addresses, in order to eliminate near-duplicates. In both of these applications, the strict global left-right ordering imposed by $O(n^2)$ time "edit distance" methods can be problematic. On the other hand, very local left-right order preservation seems to be an important part of similarity perception in humans. One simple adaptation of our σ-distance methods which goes a long way toward capturing this characteristic consists of extending the alphabet beyond single symbols to include digraphs or multigraphs. The result is increased sensitivity to local permutation. Another effective alphabet extension technique involves the addition of *feature symbols* to the alphabet to mark events such as likely phonetic transitions. We expect that the use of general concave-down distance functions (as opposed to $f(x) = x$) will improve the quality of the similarity judgments possible within the σ-distance framework.

The development above considers strings of equal length only. The unequal length case is not a difficult generalization but considering it does highlight the issue of *embedding*. By this we mean that it is implicit in our formulation that the two strings are in a sense embedded into the real line. The particular, rather natural embedding we've assumed so far maps a_i and b_i to value i on the real line, but others are possible.

A detailed comparison of our methods with "edit distance" approaches is beyond the scope of this paper. But we must point out that the "edit distance" formulation is in several senses richer than ours. First, the cost of matching different alphabet members need not be fixed. Also, our distance formulation depends on a designated embedding while the "edit distance" method requires no such specification. Finally, for some problems left-right order preservation may be desirable. On the other hand, even the simplest "edit distance" approach is $O(n^2)$ compared with the $O(n)$ or $O(n \log n)$ complexity of our method. We therefore feel that additional work

is needed to better understand the applications of our approach—and perhaps extend it.

Acknowledgments. We wish to thank Dina Kravets, Dave Robinson, and Warren Smith for helpful discussions and Dave Robinson and Kirk Kanzelberger for implementing and testing the algorithms described above.

REFERENCES

[1] A. Aggarwal, A. Bar-Noy, S. Khuller, D. Kravets, and B. Schieber, *Efficient minimum cost matching using quadrangle inequality*, in Proc. 33rd Annual IEEE Symp. on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 583–592.

[2] A. Aggarwal and M. Klawe, *Applications of generalized matrix searching to geometric algorithms*, Discrete Appl. Math., 27 (1990), pp. 3–23.

[3] A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilber, *Geometric applications of a matrix-searching algorithm*, Algorithmica, 2 (1987), pp. 195–208.

[4] A. Aggarwal and J. Park, *Notes on searching in multidimensional monotone arrays*, in Proc. 29th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 497–512.

[5] S. R. Buss, K. G. Kanzelberger, D. Robinson, and P. N. Yianilos, *Solving the Minimum-cost Matching Problem for Quasi-convex Tours: An Efficient ANSI-C Implementation*, Tech. Report CS94-370, University of California, San Diego, 1994.

[6] S. R. Buss and P. N. Yianilos, *A Bipartite Matching Approach to Approximate String Comparison and Search*, Tech. Report, NEC Research Institute, Princeton, NJ, 1995.

[7] D. Eppstein, *Sequence comparison with mixed convex and concave costs*, J. Algorithms, 11 (1990), pp. 85–101.

[8] Z. Galil and R. Giancarlo, *Speeding up dynamic programming with applications to molecular biology*, Theoret. Comput. Sci., 64 (1989), pp. 107–118.

[9] Z. Galil and K. Park, *A linear-time algorithm for concave one-dimensional dynamic programming*, Inform. Process. Lett., 33 (1990), pp. 309–311.

[10] P. Gilmore and R. Gomory, *Sequencing a one state-variable machine: A solvable case of the traveling salesman problem*, Oper. Res., 12 (1964), pp. 655–679.

[11] X. He, *An efficient parallel algorithm for finding minimum weight matching for points on a convex polygon*, Inform. Process. Lett., 37 (1991), pp. 111–116.

[12] D. S. Hirschberg and L. L. Larmore, *The least weight subsequence problem*, SIAM J. Comput., 16 (1987), pp. 628–638.

[13] A. J. Hoffman, *On simple linear programming problems*, in Convexity: Proceedings of the Seventh Symposium in Pure Mathematics of the AMS, V. Klee, ed., American Mathematical Society, Providence, RI, 1963, pp. 317–327.

[14] R. M. Karp and S.-Y. R. Li, *Two special cases of the assignment problem*, Discrete Math., 13 (1975), pp. 129–142.

[15] M. M. Klawe and D. J. Kleitman, *An almost linear time algorithm for generalized matrix searching*, SIAM J. Discrete Math., 3 (1990), pp. 81–97.

[16] D. Kravets and J. K. Park, *Selection and sorting in totally monotone arrays*, Math. Systems Theory, 24 (1991), pp. 201–220.

[17] L. L. Larmore and B. Schieber, *On-line dynamic programming with applications to the prediction of RNA secondary structure*, J. Algorithms, 12 (1991), pp. 490–515.

[18] E. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[19] Y. Mansour, J. K. Park, B. Schieber, and S. Sen, *Improved selection in totally monotone arrays*, Internat. J. Comput. Geom. Appl., 3 (1993), pp. 115–132.

[20] O. Marcotte and S. Suri, *Fast matching algorithms for points on a polygon*, SIAM J. Comput., 20 (1991), pp. 405–422.

[21] D. Sankoff and J. B. Kruskal, *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison*, Addison–Wesley, Reading, MA, 1983.

[22] P. M. Vaidya, *Geometry helps in matching*, SIAM J. Comput., 18 (1989), pp. 1201–1225.

[23] M. Werman, S. Peleg, R. Melter, and T. Kong, *Bipartite graph matching for points on a line or a circle*, J. Algorithms, 7 (1986), pp. 277–284.

[24] R. Wilber, *The concave least-weight subsequence problem revisited*, J. Algorithms, 9 (1988), pp. 418–425.

[25] F. F. Yao, *Speed-up in dynamic programming*, SIAM J. Alg. Discrete Methods, 3 (1982), pp. 523–540.
[26] P. N. Yianilos, *The Definition, Computation and Application of Symbol String Similarity Functions*, Master's thesis, Emory University, Atlanta, GA, 1978.
[27] P. N. Yianilos and S. R. Buss, *Associative memory circuit system and method, continuation-in-part*, U.S. Patent 4490811, 1984.

SIAM J. COMPUT.
Vol. 27, No. 1, pp. 202–229, February 1998
© 1998 Society for Industrial and Applied Mathematics
009

SPACE-EFFICIENT SCHEDULING OF MULTITHREADED COMPUTATIONS*

ROBERT D. BLUMOFE† AND CHARLES E. LEISERSON‡

Abstract. This paper considers the problem of scheduling dynamic parallel computations to achieve linear speedup without using significantly more space per processor than that required for a single-processor execution. Utilizing a new graph-theoretic model of multithreaded computation, execution efficiency is quantified by three important measures: T_1 is the time required for executing the computation on a 1 processor, T_∞ is the time required by an infinite number of processors, and S_1 is the space required to execute the computation on a 1 processor. A computation executed on P processors is time-efficient if the time is $O(T_1/P + T_\infty)$, that is, it achieves linear speedup when $P = O(T_1/T_\infty)$, and it is space-efficient if it uses $O(S_1 P)$ total space, that is, the space per processor is within a constant factor of that required for a 1-processor execution.

The first result derived from this model shows that there exist multithreaded computations such that no execution schedule can simultaneously achieve efficient time and efficient space. But by restricting attention to "strict" computations—those in which all arguments to a procedure must be available before the procedure can be invoked—much more positive results are obtainable. Specifically, for any strict multithreaded computation, a simple online algorithm can compute a schedule that is both time-efficient and space-efficient. Unfortunately, because the algorithm uses a global queue, the overhead of computing the schedule can be substantial. This problem is overcome by a decentralized algorithm that can compute and execute a P-processor schedule online in expected time $O(T_1/P + T_\infty \lg P)$ and worst-case space $O(S_1 P \lg P)$, including overhead costs.

Key words. parallel computing, multithreaded computing, parallel algorithms, scheduling algorithms, randomized algorithms, strict execution, stack memory

AMS subject classifications. 68Q22, 68Q25, 68M20

PII. S0097539793259471

1. Introduction.

In the course of investigating schemes for general-purpose MIMD-style parallel computation, many diverse research groups have agreed on multithreading as a dominant paradigm. As an example, modern dataflow systems [16, 19, 25, 33, 34, 35, 40, 41] partition the dataflow instructions into fixed groups called threads and arrange the instructions of each thread into a fixed sequential order at compile time. At run time, a scheduler dynamically orders execution of the threads. Other systems employ schedulers that dynamically order threads based on the availability of data in shared-memory multiprocessors [1, 10, 23] or message arrivals in message-passing multicomputers [2, 17, 29, 44].

Rapid execution of a multithreaded computation on a parallel computer requires exposing and exploiting parallelism in the computation by keeping enough threads concurrently alive to keep the processors of the computer busy. If processors are busy most of the time, the P-processor execution schedule $\mathcal{X}$ of the computation exhibits

*Received by the editors December 9, 1993; accepted for publication (in revised form) January 12, 1996. This research was supported in part by the Defense Advanced Research Projects Agency under grant N00014-91-J-1698. An extended abstract of this paper appeared in the *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing (STOC)*, San Diego, CA, ACM, New York, 1993, pp. 362–371.

http://www.siam.org/journals/sicomp/27-1/25947.html

†Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712-1188 (rdb@cs.utexas.edu). This research was conducted at the MIT Laboratory for Computer Science with additional support from a National Science Foundation Graduate Fellowship.

‡MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139 (cel@mit.edu).

linear speedup: the running time $T(\mathcal{X})$ is order P times faster than the optimal running time T_1 with 1 processor, that is, $T(\mathcal{X}) = O(T_1/P)$.

In attempting to expose parallelism, however, schedulers often end up exposing more parallelism than the computer can actually exploit, and since each living thread requires the use of a certain amount of memory, such schedulers can easily overrun the memory capacity of the machine [15, 22, 24, 39, 43]. To date, the space requirements of multithreaded computations have been managed with heuristics or not at all [14, 15, 22, 24, 26, 32, 39, 43]. In this paper, we use algorithmic techniques to address the problem of managing storage for multithreaded computations. Our goal is to develop scheduling algorithms that expose sufficient parallelism to obtain linear speedup but without exposing so much parallelism that the space requirements become excessive.

We compare the total space $S(\mathcal{X})$ required by a P-processor execution schedule $\mathcal{X}$ with the space S_1 used by a space-optimal 1-processor execution. We wish to use as little space as possible, and we argue that a space-efficient P-processor execution schedule $\mathcal{X}$ exhibits at most linear expansion of space, that is, $S(\mathcal{X}) = O(S_1 P)$.

Our first result shows that, in general, it is not possible to achieve both linear speedup and linear expansion of space. We exhibit a multithreaded computation such that any execution schedule $\mathcal{X}$ that achieves a factor of ρ speedup, that is, execution time $T(\mathcal{X}) \leq T_1/\rho$, must use space at least $S(\mathcal{X}) \geq (1/4)(\rho - 1)\sqrt{T_1} + S_1$. For such a computation, even achieving a factor of 2 speedup ($\rho = 2$) requires space that grows as a function of the serial execution time.

In order to cope with this negative result, we restrict our attention to the class of "strict" multithreaded computations. Intuitively, a strict computation is one in which no subroutine is called until all its parameters are available, although the parameters may be evaluated in parallel. Computations such as parallel divide-and-conquer, backtrack search, branch-and-bound, and game-tree search are all strict.

We show that for any strict multithreaded computation and any number P of processors, there exists a P-processor execution schedule $\mathcal{X}$ that achieves time $T(\mathcal{X}) \leq T_1/P + T_\infty$, where T_∞ is the optimal execution time on an infinite number of processors, and space $S(\mathcal{X}) \leq S_1 P$. Such a schedule exhibits linear expansion of space and linear speedup, $T(\mathcal{X}) = O(T_1/P)$, provided the average available parallelism, which we define as T_1/T_∞, is at least proportional to P, that is, $T_1/T_\infty = \Omega(P)$. We prove such schedules exist by exhibiting a simple centralized algorithm to compute them. We give a second, somewhat more efficient algorithm that computes equally good execution schedules; this algorithm is online and should be practical for moderate numbers of processors, but its use of a centralized queue makes it inefficient for large numbers of processors.

To demonstrate an algorithm that is efficient even for large machines, we give a randomized, distributed, and online scheduling algorithm that achieves space expansion proportional to $P \lg P$ for any strict computation and linear expected speedup for any strict computation with average available parallelism $T_1/T_\infty = \Omega(P \lg P)$.

We also show that some nonstrictness can be allowed in an otherwise strict computation in a way that may improve performance but does not adversely affect the time and space bounds.

The remainder of this paper is organized as follows. Section 2 develops a formal model of multithreaded computation and execution schedules. In section 3, we characterize multithreaded computations with three parameters and state some basic bounds relating these parameters to execution time and space. The lower bound for general multithreaded computations is presented in section 4. The upper bound

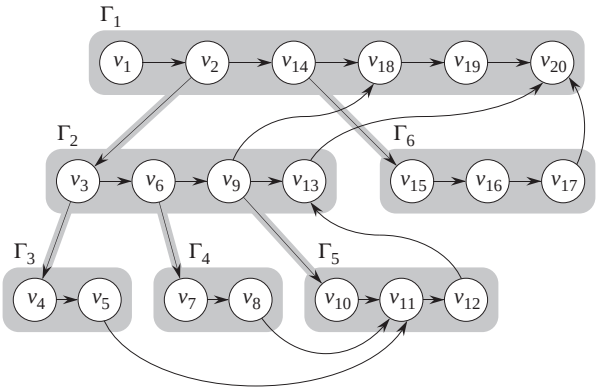


FIG. 2.1. *A multithreaded computation. This computation contains* 20 *tasks* $v_1, v_2, \ldots, v_{20}$ *and six threads* $\Gamma_1, \Gamma_2, \ldots, \Gamma_6$.

for strict computations and the technique for handling limited nonstrictness are presented in section 5. Section 6 presents a distributed scheduling algorithm for strict computations. Finally, in section 7 we conclude with a discussion of related and future work.

2. A model for multithreaded computation. This section defines the model of multithreaded computation that we use in this paper. We also define what it means for a parallel computer to execute a multithreaded computation.

A multithreaded computation is composed of a set of threads, each of which is a sequential ordering of unit-size tasks. In Figure 2.1, for example, each shaded block is a thread with circles representing tasks and the horizontal edges, called *continue* edges, representing the sequential ordering. Thread Γ_5 of this example contains three tasks: v_{10}, v_{11}, and v_{12}. The tasks of a thread must execute in this sequential order from the first (leftmost) task to the last (rightmost) task. In order to execute a thread, we allocate for it a chunk of memory, called an *activation frame*, that the tasks of the thread can use to store the values on which they compute.

A P-processor *execution schedule* for a multithreaded computation determines which processors of a P-processor parallel computer execute which tasks at each step. In any given step of an execution schedule, each processor either executes a single task or sits idle. A 3-processor execution schedule for our example computation (Figure 2.1) is shown in Figure 2.2. At step 3 of this example, processors p_1 and p_2 each execute a task while processor p_3 sits idle.

During the course of its execution, a thread may create, or *spawn*, other threads. Spawning a thread is like a subroutine call except that the spawning thread can operate concurrently with the spawned thread. We consider spawned threads to be children of the thread that did the spawning, and a thread may spawn as many children as it desires. In this way, threads are organized into a *spawn tree* as indicated in Figure 2.1 by the downward-pointing, shaded edges, called *spawn* edges, that connect threads to their spawned children. The spawn tree is the parallel analogue of a call tree. In our example computation, the spawn tree's *root* thread Γ_1 has two children, Γ_2 and Γ_6, and thread Γ_2 has three children, Γ_3, Γ_4, and Γ_5. Threads Γ_3, Γ_4, Γ_5, and Γ_6, which have no children, are *leaf* threads.

Each spawn edge goes from a specific task—the task that actually does the spawn operation—in the parent thread to the first task of the child thread. An execution

step	living threads						processor activity p_1	p_2	p_3
1	$\mathbf{\Gamma_1}$						v_1		
2	$\mathbf{\Gamma_1}$						v_2		
3	$\mathbf{\Gamma_1}$	$\mathbf{\Gamma_2}$					v_3	v_{14}	
4	Γ_1	$\mathbf{\Gamma_2}$	$\mathbf{\Gamma_3}$			$\mathbf{\Gamma_6}$	v_4	v_6	v_{15}
5	Γ_1	$\mathbf{\Gamma_2}$	$\mathbf{\Gamma_3}$	$\mathbf{\Gamma_4}$		$\mathbf{\Gamma_6}$	v_5	v_9	v_{16}
6	$\mathbf{\Gamma_1}$	Γ_2		$\mathbf{\Gamma_4}$	$\mathbf{\Gamma_5}$	$\mathbf{\Gamma_6}$	v_7	v_{10}	v_{17}
7	$\mathbf{\Gamma_1}$	Γ_2		$\mathbf{\Gamma_4}$	Γ_5		v_8	v_{18}	
8	$\mathbf{\Gamma_1}$	Γ_2			$\mathbf{\Gamma_5}$			v_{19}	v_{11}
9	Γ_1	Γ_2			$\mathbf{\Gamma_5}$				v_{12}
10	Γ_1	$\mathbf{\Gamma_2}$							v_{13}
11	$\mathbf{\Gamma_1}$								v_{20}

FIG. 2.2. *A 3-processor execution schedule for the computation of Figure* 2.1. *This schedule lists the living threads at the start of each step and the task (if any) executed by each of the three processors, p_1, p_2, and p_3, at each step. Living threads that are ready are listed in bold. The other living threads are stalled.*

schedule must obey this edge in that no processor may execute a task in a spawned child thread until after the spawning task in the parent thread has been executed. In our example computation (Figure 2.1), due to the spawn edge (v_6, v_7), task v_7 cannot be executed until after the spawning task v_6. Consistent with our unit-time model of tasks, a single task may spawn at most one child. When the spawning task executes, it allocates an activation frame for the new child thread. Once a thread has been spawned and its frame has been allocated, we say the thread is *alive* or *living*. When the last task of a thread executes, it deallocates its frame and the thread *dies*. In our 3-processor execution schedule (Figure 2.2), thread Γ_5 is spawned at step 5 and dies at step 9. Therefore, it is living at steps 6, 7, 8, and 9.

An execution schedule must respect one more kind of dependency. Consider a task that produces a data value that is consumed by another task. Such a producer/consumer relationship precludes the consuming task from executing until after the producing task. To enforce such orderings, we introduce *data-dependency* edges, as shown in Figure 2.1 by the curved edges. If the execution of a thread arrives at a consuming task before the producing task has executed, execution of the consuming thread cannot continue; the thread *stalls*. Once the producing task executes, the data dependency is *resolved*, which *enables* the consuming thread to resume with its execution; the thread becomes *ready*. For example, at step 4 of our 3-processor execution schedule (Figure 2.2), thread Γ_1 is stalled at task v_{18} because task v_9 has not yet been executed. At step 5 task v_9 is executed by processor p_2, thereby enabling thread Γ_1. At step 6, thread Γ_1 is ready at task v_{18}. A multithreaded computation does not model the mechanism by which data dependencies get resolved or unresolved dependencies get detected.

An execution schedule must obey the constraints given by the data-dependency, spawn, and continue edges of the computation. These edges form a directed graph of tasks, and no processor may execute a task until after all of the task's predecessors in this graph have been executed. So that execution schedules exist, this graph must be acyclic. That is, it must be a directed acyclic graph, or *dag*. At any given step of an execution schedule, a task is *ready* if all of its predecessors in the dag have been executed. Only ready tasks may be executed.

We make the simplifying assumption that a parent thread remains alive until all its children die, and thus, a thread does not deallocate its activation frame until all its children's frames have been deallocated. Although this assumption is not strictly necessary, it gives the execution a natural structure, and it will simplify our analyses of space utilization. We also assume that the frames hold all the values used by the computation; there is no global storage available to the computation outside the frames. Therefore, the space used at a given time in executing a computation is the total size of all frames used by all living threads at that time, and the total space used in executing a computation is the maximum such value over the course of the execution.

To summarize, a multithreaded computation can be viewed as a dag of tasks connected by continue, spawn, and data-dependency edges. The tasks are connected by continue edges into threads, and the threads form a spawn tree with the spawn edges. When a thread is spawned, an activation frame is allocated and this frame remains allocated as long as the thread remains alive. A living thread may be either ready or stalled due to an unresolved data dependency.

The notion of an execution schedule is independent of any real machine characteristics. An execution schedule simply requires that no processor executes more than one task per time step and every task is executed at a time step after all of its predecessor tasks (which connect to it via continue, spawn, or data-dependency edges) have been executed. A given execution schedule may not be viable for a real machine, since the schedule may not account for properties such as communication latency. For example, in our 3-processor execution schedule (Figure 2.2), task v_{11} is executed at step 8 by processor p_3 exactly one step after v_8 is executed by processor p_1, even though there is a data dependency between them that surely requires some latency to be resolved.

It is important to note the difference between what we are calling a multithreaded computation and a program. A multithreaded computation is the "parallel task stream" resulting from the execution of a multithreaded program with a given set of inputs. Unlike a serial computation in which the task stream is totally ordered, a multithreaded computation only partially orders its tasks. In general, a multithreaded computation is not a statically determined object; rather, the computation unfolds dynamically during execution as determined by the program and the input data. For example, a program may have conditionals, and therefore, the order of tasks (or even the set of tasks) executed in a thread may not be known until the thread is actually executed. We can think of a multithreaded computation as encapsulating both the program and the input data. The computation then reveals itself dynamically during execution.

3. Time and space. We shall characterize the time and space of an execution of a multithreaded computation in terms of three fundamental parameters: work, computation depth, and activation depth. We first introduce work and computation depth, which relate to the execution time, and then we focus on activation depth, which relates to the storage requirements.

The two time parameters are based on the underlying graph structure of the multithreaded computation. If we ignore the shading in Figure 2.1 that organizes tasks into threads, our multithreaded computation is just a dag of tasks. We define the *work* of the computation to be the total number of tasks and the *computation depth* to be the length of a longest directed path in the dag.

We quantify and bound the execution time of a computation on a P-processor

parallel computer in terms of the computation's work and depth. For a given computation, let $T(\mathcal{X})$ denote the time to execute the computation using P-processor execution schedule $\mathcal{X}$, and let

$$T_P = \min_{\mathcal{X}} T(\mathcal{X})$$

denote the minimum execution time with P processors—the minimum being taken over all P-processor execution schedules for the computation. Then T_1 is the work of the computation, since a 1-processor computer can only execute one task at each step, and T_∞ is the computation depth, since, even with arbitrarily many processors, each task on a path must execute serially.

Still viewing the computation as a dag, we borrow some basic results on dag scheduling to bound T_P. A computer with P processors can execute at most P tasks per step, and since the computation has T_1 tasks, we have $T_P \geq T_1/P$. Of course, we also have $T_P \geq T_\infty$. Early work by Graham [20, 21] and independently by Brent [11, Lemma 2] yields the bound $T_P \leq T_1/P + T_\infty$. The following theorem extends these results minimally to show that this upper bound on T_P can be obtained by *greedy schedules*, i.e., those in which at each step of the execution, if at least P tasks are ready, then P tasks execute, and if fewer than P tasks are ready, then all execute.

THEOREM 3.1 (the greedy-scheduling theorem). *For any multithreaded computation with work T_1 and computation depth T_∞, and for any number P of processors, every greedy P-processor execution schedule $\mathcal{X}$ achieves $T(\mathcal{X}) \leq T_1/P + T_\infty$.*

Proof. Let $G = (V, E)$ denote the underlying dag of the computation. Thus we have $|V| = T_1$, and a longest directed path in G has length T_∞. Consider a greedy P-processor execution schedule $\mathcal{X}$ where the set of tasks executed at time i, for $i = 1, 2, \ldots, k$, is denoted $\mathcal{V}_i$, with $k = T(\mathcal{X})$. The $\mathcal{V}_i$ form a partition of V.

We shall consider the progression $\langle G_0, G_1, G_2, \ldots, G_k \rangle$ of dags, where $G_0 = G$, and for $i = 1, 2, \ldots, k$, we have $V_i = V_{i-1} - \mathcal{V}_i$, and G_i is the subgraph of G_{i-1} induced by V_i. In other words, G_i is obtained from G_{i-1} by removing from G_{i-1} all the tasks that are executed by $\mathcal{X}$ at step i and all edges incident on these tasks. We shall show that each step of the execution either decreases the size of the dag or decreases the length of the longest path in the dag.

We account for each step i according to $|\mathcal{V}_i|$. Consider a step i with $|\mathcal{V}_i| = P$. In this case, $|V_i| = |V_{i-1}| - P$, so since $|V| = T_1$, there can be at most $\lfloor T_1/P \rfloor$ such steps. Now, consider a step i with $|\mathcal{V}_i| < P$. In this case, since $\mathcal{X}$ is greedy, $\mathcal{V}_i$ must contain every vertex of G_{i-1} with in-degree 0. Therefore, the length of a longest path in G_i is one less than the length of a longest path in G_{i-1}. Since the length of a longest path in G is T_∞, there can be no more than T_∞ steps i with $|\mathcal{V}_i| < P$.

Consequently, the time it takes schedule $\mathcal{X}$ to execute the computation is $T(\mathcal{X}) \leq \lfloor T_1/P \rfloor + T_\infty \leq T_1/P + T_\infty$. □

The greedy-scheduling theorem (Theorem 3.1) can be interpreted in two important ways. First, the time bound given by the theorem says that any greedy schedule yields an execution time that is within a factor of 2 of an optimal schedule, which follows because $T_1/P + T_\infty \leq 2\max\{T_1/P, T_\infty\}$ and $T_P \geq \max\{T_1/P, T_\infty\}$. Second, the greedy-scheduling theorem tells us when we can obtain *linear parallel speedup*, that is, when we can find an execution schedule $\mathcal{X}$ such that $T(\mathcal{X}) = \Theta(T_1/P)$. Specifically, when the number P of processors is no more than the *average available parallelism* T_1/T_∞, then $T_1/P \geq T_\infty$, which implies that for a greedy schedule $\mathcal{X}$, we have $T(\mathcal{X}) \leq 2T_1/P$. We shall be especially interested in the regime where $P = O(T_1/T_\infty)$

and linear speedup is possible, since outside this regime, linear speedup is impossible to achieve because $T_P \geq T_\infty$.

These results on dag scheduling have been known for years. A multithreaded computation, however, adds further structure to the dag: the partitioning of tasks into threads. This additional structure allows us to quantify the space used in executing a multithreaded computation. Once we have quantified space usage, we will look back at the greedy-scheduling theorem and consider whether there exist execution schedules that achieve similar time bounds while also making efficient use of space. Of course, we will have to quantify a space bound to capture what we mean by "efficient use of space."

We shall focus on a space parameter for a multithreaded computation which is based on the tree structure of threads. If we collapse each thread into a single node and consider just the spawn edges, the multithreaded computation is just a spawn tree of threads. We define the *activation depth* of a thread to be the sum of the sizes of the activation frames of all its ancestors, including itself. The *activation depth* of a multithreaded computation is the maximum activation depth of any thread.

We shall denote the space required by a P-processor execution schedule $\mathcal{X}$ of a multithreaded computation by $S(\mathcal{X})$. Recall that $S(\mathcal{X})$ is just the maximum, over all steps in $\mathcal{X}$, of the sum of the sizes of the activation frames of the living threads at that step. Since we can always simulate a P-processor execution with a 1-processor execution that uses no more space, we have $S_1 \leq S(\mathcal{X})$, where $S_1 = \min_{\mathcal{X}} S(\mathcal{X})$ denotes the minimum space used by a 1-processor execution.

The following simple theorem shows that the activation depth of a computation is a lower bound on the space required to execute it.

THEOREM 3.2. *Let $\mathcal{A}$ be the activation depth of a multithreaded computation, and let $\mathcal{X}$ be a P-processor execution schedule of the computation. Then we have $S(\mathcal{X}) \geq \mathcal{A}$, and more specifically, we have $S_1 \geq \mathcal{A}$.*

Proof. In any schedule, the leaf thread with greatest activation depth must be alive at some time step. Since we assume that if a thread is alive, its parent is alive, when the deepest leaf thread is alive, all its ancestors are alive, and hence, all its ancestors' frames are allocated. However, the sum of the sizes of its ancestors' activation frames is just the activation depth. Since $S(\mathcal{X}) \geq \mathcal{A}$ holds for all P-processor schedules $\mathcal{X}$ and all P, it holds for the minimum-space execution schedule, and hence, $S_1 \geq \mathcal{A}$. □

Given the lower bound of activation depth on the space used by a P-processor schedule, it is natural to ask whether the activation depth can be achieved as an upper bound. In general, the answer is no, since all the threads in a computation may contain a cycle of data dependencies that force all of them to be simultaneously living in any execution schedule. For the class of "depth-first" computations, however, space equal to the activation depth can be achieved by a 1-processor schedule.

A *depth-first* computation is a multithreaded computation in which a left-to-right depth-first search of tasks in the spawn tree always visits all the tasks on which a given task depends before it visits the given task. In the example computation of Figure 2.1, the left-to-right depth-first search order is $v_1, v_2, \ldots, v_{20}$, and this computation is depth-first. In fact, this depth-first search produces a 1-processor execution schedule which is just the familiar stack-based execution: the serial depth-first execution begins with the root thread and executes its tasks until it either spawns a child thread or dies. If the thread spawns a child, the parent thread is put aside to be resumed only after the child thread dies; the scheduler then begins work on the child, executing the

child until it either spawns a child or dies.

THEOREM 3.3. *For any depth-first computation, we have* $S_1 = \mathcal{A}$.

Proof. At any time in a serial depth-first execution of the computation, the set of living threads always forms a path from the root. Therefore, the space required is just the activation depth of the computation. By Theorem 3.2, $S_1 \geq \mathcal{A}$, and thus the the space used is the minimum possible. □

The remainder of this paper considers only depth-first computations, and we shall use S_1 to denote a computation's activation depth.

We now turn our attention to determining how much space $S(\mathcal{X})$ a P-processor execution schedule $\mathcal{X}$ can use and still be considered efficient with respect to space usage. Our strategy is to compare the space used by a P-processor schedule with the space required by an optimal 1-processor schedule. Of course, we can always ignore $P - 1$ of the processors to match the single-processor space bounds, and therefore, our goal is to use small space while obtaining linear speedup.

Even for depth-first computations, a P-processor schedule may use nearly P times the space of a 1-processor schedule. Consider, for example, a computation in which the root thread is a loop that spawns a child thread for each iteration. A single processor executing this computation uses only the space needed for a single iteration (plus the space used by the root), since upon completion of an iteration, all the memory can be freed and then reused for the next iteration. A natural P-processor execution, however, might execute P iterations concurrently, thereby requiring the memory of P iterations. Such a P-processor execution schedule $\mathcal{X}$ uses space $S(\mathcal{X}) = \Theta(S_1 P)$.

In fact, a P-processor schedule that uses only P times the space of a single processor is arguably efficient, since on average, each of the P processors only needs as much memory as is used by the 1 processor. We would, of course, like to do better, but an expansion in space that is linear in the number of processors, while achieving linear speedup, is quite good, since the time-space product is bounded by a value independent of P:

$$\begin{aligned} T(\mathcal{X})S(\mathcal{X}) &= O(T_1/P) \cdot O(S_1 P) \\ &= O(T_1 S_1) \ . \end{aligned}$$

We shall show in section 4 that achieving linear speedup and linear expansion of space simultaneously is impossible in general, even for depth-first computations. For the class of strict computations, however, section 5 shows that one can achieve both.

4. Lower bound. In this section we show that there exist multithreaded computations for which no execution schedule can achieve both linear speedup and linear expansion of space. In particular, for any amount of serial space S_1 and any (reasonably large) serial execution time T_1, we can exhibit a depth-first multithreaded computation with work T_1 and activation depth S_1 but with provably bad time/space tradeoff characteristics. Being depth-first, we know from Theorem 3.3 that our computation can be executed using serial space S_1. Furthermore, we know from the greedy-scheduling theorem (Theorem 3.1) that for any number P of processors, any greedy P-processor execution schedule $\mathcal{X}$ achieves $T(\mathcal{X}) \leq T_1/P + T_\infty$. Our computation has computation depth $T_\infty \approx \sqrt{T_1}$, and consequently, for $P = O(\sqrt{T_1})$, a greedy P-processor schedule $\mathcal{X}$ yields $T(\mathcal{X}) = O(T_1/P)$, i.e., linear speedup. We show, however, that any P-processor schedule $\mathcal{X}$ achieving $T(\mathcal{X}) = O(T_1/P)$ must use space $S(\mathcal{X}) = \Omega(\sqrt{T_1}(P-1))$. Of course, $\sqrt{T_1}$ may be much larger than S_1, and hence, this space bound is nowhere near linear in its space expansion.

THEOREM 4.1. *For any amount of serial space $S_1 \geq 4$ and serial time $T_1 \geq 16S_1^2$, there exists a depth-first multithreaded computation with work T_1, computation depth $T_\infty \leq 8\sqrt{T_1}$, and activation depth S_1 with the following property: for any number P of processors and any value ρ in the range $1 \leq \rho \leq (1/8)T_1/T_\infty$, if $\mathcal{X}$ is a P-processor execution schedule that achieves speedup ρ—that is, $T(\mathcal{X}) \leq T_1/\rho$— then the schedule requires space $S(\mathcal{X}) \geq (1/4)(\rho - 1)\sqrt{T_1} + S_1$.*

Proof. To exhibit a depth-first multithreaded computation with work T_1, computation depth T_∞, and activation depth S_1, we first ignore the partitioning of tasks into threads and consider just the dag structure of the computation. Minus a few tasks and dependencies, the dag appears as in Figure 4.1(a). The tasks are organized into

$$m = \sqrt{T_1}/8$$

separate components $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_{m-1}$ that we call *chains*.[1] Each chain begins with

$$\lambda = \sqrt{T_1}/S_1$$

tasks that we call *headers* (vertical hashed in Figure 4.1(a)). After the headers, each chain contains

$$\nu = 6\sqrt{T_1}$$

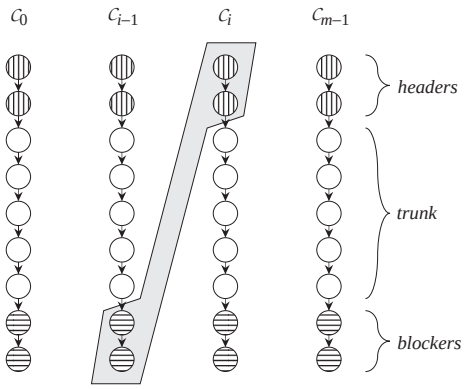
tasks (plain white in Figure 4.1(a)) that form the *trunk*. At the end of each chain, there are λ *blockers* (horizontal hashed in Figure 4.1(a)). Each chain, therefore, consists of $2\lambda + \nu = 2(\sqrt{T_1}/S_1) + 6\sqrt{T_1}$ tasks. Since there are $m = \sqrt{T_1}/8$ chains, the total number of tasks accounted for by the m chains is $(2\sqrt{T_1}/S_1 + 6\sqrt{T_1})\sqrt{T_1}/8 = (3/4)T_1 + (1/4)T_1/S_1$, and this number is no more than $(13/16)T_1$ since $S_1 \geq 4$. The remaining (at least) $(3/16)T_1$ tasks form the parts of the computation not shown in Figure 4.1(a).

There are no dependencies between different chains, so the average available parallelism T_1/T_∞ is at least $m = \sqrt{T_1}/8$ and the computation depth T_∞ is no more than $8\sqrt{T_1}$ as promised.

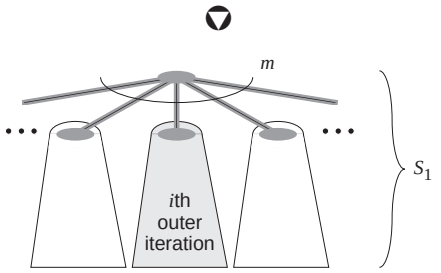
Now, consider the partitioning of the tasks from each chain into the actual threads. As alluded to in Figure 4.1(b), the root thread has m child threads, each of which is the root of a subcomputation that we call an *outer iteration*. (The outer iterations contain *inner iterations* that will be discussed later.) Each of these outer iterations contains $\sqrt{T_1}/2$ threads. As illustrated by the shading in Figure 4.1, the ith outer iteration for $i = 1, 2, \ldots, m-1$ contains both the header tasks of chain $\mathcal{C}_i$ and the blocker tasks of chain $\mathcal{C}_{i-1}$. These tasks are organized into the threads of the outer iteration so as to ensure that chain $\mathcal{C}_i$ cannot begin executing its trunk tasks until all $\sqrt{T_1}/2$ of the outer iteration's threads have been spawned, and none of these threads can die until chain $\mathcal{C}_{i-1}$ begins executing its blocker tasks. (We will exhibit this organization later.) Thus, if chain $\mathcal{C}_i$ begins executing its trunk tasks before chain $\mathcal{C}_{i-1}$ finishes its, then the execution will require at least $\sqrt{T_1}/2$ space.

For any number P of processors, consider any valid P-processor execution schedule $\mathcal{X}$. For each chain $\mathcal{C}_i$, let $t_i^{(s)}$ denote the time step at which $\mathcal{X}$ executes the first

[1] In what follows, we refer to a number x of objects (such as tasks) when x may not be integral. Rounding these quantities to integers does not affect the correctness of the proof. For ease of exposition, we shall not consider the issue.



(A) *Chains of tasks.*



(B) *Outer iterations.*

FIG. 4.1. *Constructing a computation with no efficient execution schedule. The header tasks of chain $\mathcal{C}_i$ and the blocker tasks of chain $\mathcal{C}_{i-1}$ are both placed in the threads of the ith outer iteration.*

trunk task of $\mathcal{C}_i$, and let $t_i^{(\mathrm{f})}$ denote the first time step at which $\mathcal{X}$ executes a blocker task of $\mathcal{C}_i$. Since the trunk has length ν and no blocker task of $\mathcal{C}_i$ can execute until after the last trunk task of $\mathcal{C}_i$, we have $t_i^{(\mathrm{f})} - t_i^{(\mathrm{s})} \geq \nu$.

Now consider two chains, $\mathcal{C}_i$ and $\mathcal{C}_{i-1}$, and suppose $t_i^{(\mathrm{s})} < t_{i-1}^{(\mathrm{f})}$; this is the scenario we described as using at least $\sqrt{T_1}/2$ space. In this case, we consider the time interval from $t_i^{(\mathrm{s})}$ (inclusive) to $t_{i-1}^{(\mathrm{f})}$ (exclusive) during which we say that chain $\mathcal{C}_i$ is *exposed*, and we let $\tau_i = t_{i-1}^{(\mathrm{f})} - t_i^{(\mathrm{s})}$ denote the amount of time chain $\mathcal{C}_i$ is exposed. See Figure 4.2. If $t_i^{(\mathrm{s})} \geq t_{i-1}^{(\mathrm{f})}$ then chain $\mathcal{C}_i$ is never exposed and we let $\tau_i = 0$. As we have seen, over the time interval during which a chain is exposed, it uses at least $\sqrt{T_1}/2$ space. We will show that in order to achieve speedup ρ—that is $T(\mathcal{X}) \leq T_1/\rho$—there must be some time step during the execution at which at least $\lceil (3/4)\rho \rceil - 1$ chains are exposed.

If schedule $\mathcal{X}$ is such that $T(\mathcal{X}) \leq T_1/\rho$, then we must have $t_{m-1}^{(\mathrm{f})} - t_0^{(\mathrm{s})} \leq T_1/\rho$. We can expand this inequality to yield

$$\begin{aligned} T_1/\rho &\geq t_{m-1}^{(\mathrm{f})} - t_0^{(\mathrm{s})} \\ &= \sum_{i=0}^{m-1} \left(t_i^{(\mathrm{f})} - t_i^{(\mathrm{s})} \right) - \sum_{i=1}^{m-1} \left(t_{i-1}^{(\mathrm{f})} - t_i^{(\mathrm{s})} \right) . \end{aligned} \tag{4.1}$$

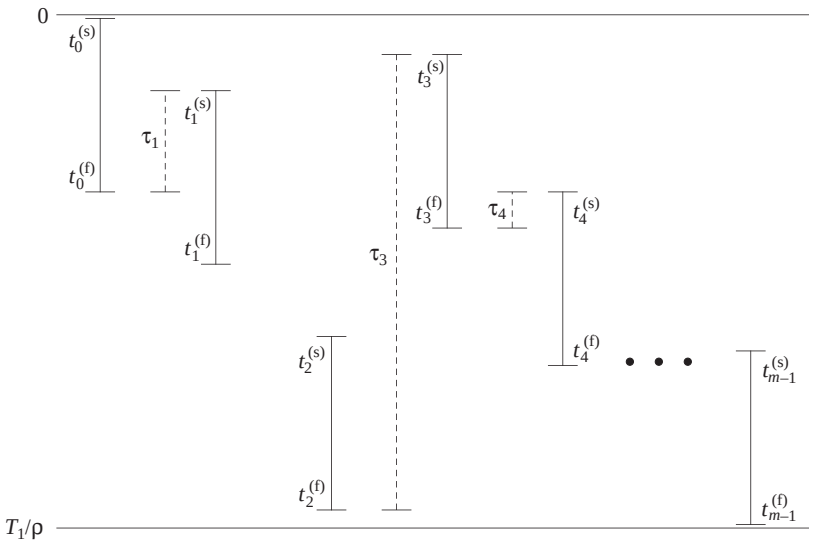


FIG. 4.2. *Scheduling the execution of the chains. A solid vertical interval from* $t_i^{(s)}$ *to* $t_i^{(f)}$ *indicates the time during which the trunk of chain* $\mathcal{C}_i$ *is being executed. When* $t_i^{(s)} < t_{i-1}^{(f)}$*, we can define an interval, shown dashed, of length* $\tau_i = t_{i-1}^{(f)} - t_i^{(s)}$*, during which chain* $\mathcal{C}_i$ *is exposed.*

Considering the first sum, we recall that $t_i^{(f)} - t_i^{(s)} \geq \nu$, hence,

$$\sum_{i=0}^{m-1} \left(t_i^{(f)} - t_i^{(s)} \right) \geq m\nu \ . \tag{4.2}$$

Considering the second sum of inequality (4.1), when $t_{i-1}^{(f)} t_i^{(s)}$ (so $\mathcal{C}_i$ is exposed), we have $\tau_i = t_{i-1}^{(f)} - t_i^{(s)}$, and otherwise, $\tau_i = 0 \geq t_{i-1}^{(f)} - t_i^{(s)}$. Therefore,

$$\sum_{i=1}^{m-1} \left(t_{i-1}^{(f)} - t_i^{(s)} \right) \leq \sum_{i=1}^{m-1} \tau_i \ . \tag{4.3}$$

Substituting inequalities (4.2) and (4.3) back into inequality (4.1), we obtain

$$\sum_{i=1}^{m-1} \tau_i \geq m\nu - T_1/\rho \ .$$

Let *exposed*(t) denote the number of chains exposed at time step t, and observe that

$$\sum_{t=1}^{T_1/\rho} \mathit{exposed}(t) = \sum_{i=i}^{m-1} \tau_i \ .$$

Then the average number of exposed chains per time step is

$$\frac{1}{T_1/\rho} \sum_{t=1}^{T_1/\rho} \mathit{exposed}(t) = \frac{1}{T_1/\rho} \sum_{i=1}^{m-1} \tau_i$$

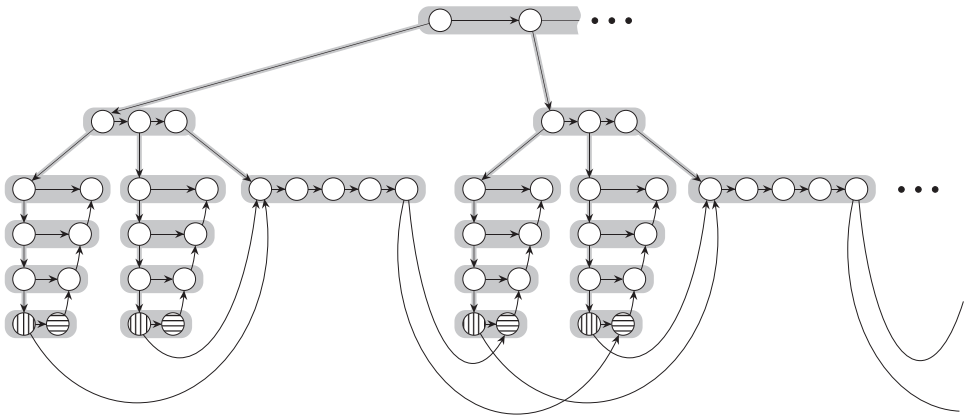


FIG. 4.3. *Laying out the chains into the threads of a multithreaded computation. As before, the header tasks are vertical hashed, and the blocker tasks are horizontal hashed. In this example, each activation frame has unit size so* $S_1 = 6$. *Also, in this example* $\lambda = 2$, $\nu = 5$, *and only the first* 2 *out of the* m *tasks in the root thread are shown. Each task of the root thread spawns a child (an outer iteration), and each child thread contains* $\lambda + 1 = 3$ *tasks; the first* λ *of these spawn a child thread which is the root of an inner iteration with activation depth* $S_1 - 2 = 4$, *and the last one spawns a leaf thread with the* $\nu = 5$ *trunk tasks of a single chain.*

$$\begin{aligned} &\geq \frac{1}{T_1/\rho}\left(m\nu - T_1/\rho\right) \\ &= \frac{3}{4}\rho - 1, \end{aligned}$$

since $m = \sqrt{T_1}/8$ and $\nu = 6\sqrt{T_1}$. There must be some time step t^* for which $\mathit{exposed}(t^*)$ is at least the average, and consequently,

$$\mathit{exposed}(t^*) \geq \left\lceil \frac{3}{4}\rho \right\rceil - 1 \ .$$

Now, recalling that each exposed chain uses space $\sqrt{T_1}/2$, we have

$$\begin{aligned} S(\mathcal{X}) &\geq \left(\left\lceil \frac{3}{4}\rho \right\rceil - 1\right)\frac{1}{2}\sqrt{T_1} \\ &\geq \frac{1}{4}(\rho - 1)\sqrt{T_1} + S_1 \end{aligned}$$

for $S_1 \leq \sqrt{T_1}/4$ (which is true since we have $T_1 \geq 16S_1^2$).

All that remains is exhibiting the organization of the tasks of each chain into a depth-first multithreaded computation with work T_1, computation depth $T_\infty \leq 8\sqrt{T_1}$, and activation depth S_1 in such a way that each exposed chain uses $\sqrt{T_1}/2$ space. There are actually many ways of creating such a computation. One such way, which uses unit-size activation frames for each thread, is shown in Figure 4.3.

For the multithreaded computation of Figure 4.3, the root thread contains m tasks, each of which spawns a child thread (an outer iteration). Each child thread contains $\lambda + 1$ tasks; the first λ of these spawn a child thread which is the root of a subcomputation that we call an *inner iteration*. Each inner iteration has activation depth $S_1 - 2 \geq S_1/2$ (since $S_1 \geq 4$), and the last one spawns a leaf thread with the ν

trunk tasks of a single chain. Each of these inner iterations contains a single header from one chain and a single blocker from the previous chain (except in the case of the first group of λ) as shown in Figure 4.3. The header and blocker in an inner iteration are organized such that in order to execute the header, all $S_1 - 2$ of the threads in the inner iteration must be spawned, and none of them can die until the blocker executes. Thus, when a chain is exposed, all λ of these inner iterations have all of their threads living, thereby using space $\lambda(S_1 - 2) \geq (\sqrt{T_1}/S_1)(S_1/2) = \sqrt{T_1}/2$.

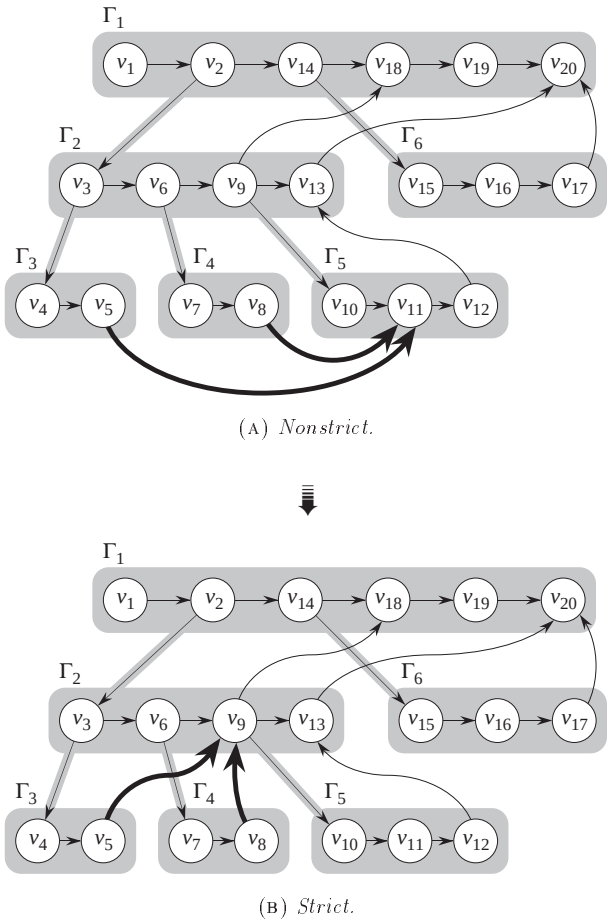
We can verify from Figure 4.3 and from the given values of m, λ, and ν that this construction actually has work slightly less than T_1; in order to make the work equal to T_1 we can just add the extra tasks evenly among the threads that contain the trunk of each chain (thereby increasing ν by a bit). Also, we can verify that $T_\infty \leq 8\sqrt{T_1}$. Finally, looking at Figure 4.3 we can see that this computation is indeed depth-first. □

The construction of a multithreaded computation with provably bad time/space characteristics as just described can be modified in various ways to accommodate various restrictions to the model while still obtaining the same result. For example, some real multithreaded systems require limits on the number of tasks in a thread, data dependencies that only go to the first task of a thread, limited fan-in for data dependencies, or a limit on the number of children a thread can have. Simple changes to the construction just described can produce multithreaded computations that accommodate any or all of these restrictions and still have the same provably bad time/space tradeoff. Thus, the lower bound of Theorem 4.1 holds even for multithreaded computations with any or all of these restrictions.

5. Scheduling algorithms for strict multithreaded computations. In the view of negative results from section 4, we consider scheduling algorithms for a specific class of depth-first multithreaded computations called "strict" computations. In this section, we show that for any strict multithreaded computation and any number P of processors, there exists a P-processor execution schedule $\mathcal{X}$ that achieves time $T(\mathcal{X}) \leq T_1/P + T_\infty$. We give two algorithms to compute such a schedule. We conclude this section by showing how some nonstrictness can be allowed in an otherwise strict computation in a way that may improve performance, but which does not adversely affect our asymptotic time and space bounds.

Given a multithreaded computation, a scheduling algorithm for a P-processor parallel computer must compute a P-processor execution schedule. In computing such a schedule, the algorithm does not know the entire computation; the computation actually unfolds dynamically during the course of execution, and consequently, the scheduling algorithm must be online. At any given time during the execution, the scheduler has a set of living threads, some of which are ready and some of which are stalled. There might be some extra information attached to each thread that the scheduling algorithm can use in deciding which ready threads get executed by which processors, but the scheduler cannot know about the structure of the portion of the computation not yet executed.

To cope with the lower bound from Theorem 4.1, we now restrict our attention to those multithreaded computations in which every data dependency goes from a thread to one of its ancestors in the spawn tree. It turns out that requiring all data dependencies to go from a thread to one of its ancestors can be viewed as requiring that all function invocations (in a functional language) be strict, and therefore, we refer to this class of computations as *strict* multithreaded computations. For example, the computation shown in Figure 5.1(a) is not strict since the bold data dependencies



(A) *Nonstrict.*

(B) *Strict.*

FIG. 5.1. (A) *This multithreaded computation (the same as Figure* 2.1*) is nonstrict since it has nonstrict data dependencies (shown bold) that go to nonancestor threads.* (B) *If we replace the nonstrict data dependencies with new strict ones (shown bold) we obtain a strict computation since all data dependencies go from a child thread to an ancestor thread.*

violate the strictness condition just stated, but by promoting these dependencies we obtain the strict computation shown in Figure 5.1(b).

Strict multithreaded computations are depth-first computations, since no data dependency can go between two distinct subcomputations of a thread. Once a thread Γ has been spawned in a strict computation, a single processor can complete the execution of Γ and all of its descendant threads by using a depth-first schedule, even if no other progress is made on other parts of the computation. In other words, from the time the thread Γ is spawned until the time Γ dies, there is always at least one thread from the subtree rooted at Γ that is ready. This property allows us to derive algorithms to schedule the execution of these computations with efficient use of both time and space.

Algorithm GDF (which stands for *global depth-first*) maintains all living threads in a global queue prioritized by activation depth, i.e., the deepest threads get highest priority. At each step of the algorithm, the scheduler removes from the queue the P deepest ready threads (if there are fewer than P ready threads, it just removes them

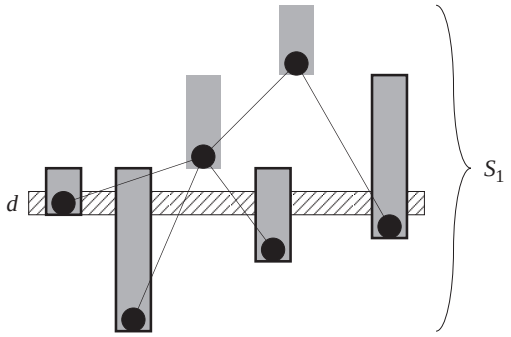


FIG. 5.2. *The spawn tree corresponding to the example computation of Figure* 2.1. *The bold outlined threads span depth d.*

all) and assigns them arbitrarily to the P processors so that each processor receives at most one thread. Each processor that has an assigned thread then executes one task from that thread. To complete the step, all surviving threads and all newly spawned threads are placed back into the global queue.

THEOREM 5.1. *For any number P of processors and any strict multithreaded computation with work T_1, computation depth T_∞, and activation depth S_1, Algorithm* GDF *computes a P-processor schedule $\mathcal{X}$ that uses space $S(\mathcal{X}) \le S_1 P$ and time $T(\mathcal{X}) \le T_1/P + T_\infty$.*

Proof. The time bound follows immediately from the greedy-scheduling theorem (Theorem 3.1), since GDF always produces a greedy schedule.

To prove the space bound, we show that the queue never contains more than P threads (ready or otherwise) that span any activation depth. A thread Γ *spans* an activation depth d, if Γ has activation depth $\mathcal{A}(\Gamma) \ge d$, and either Γ is the root or the parent thread Γ' of Γ has activation depth $\mathcal{A}(\Gamma') < d$. For example, Figure 5.2 depicts the spawn tree corresponding to the computation of Figure 2.1. Each thread has height equal to the size of its activation frame and is located so that the top of its activation frame is aligned with the bottom of its parent's activation frame. In this way, each black node is located at its thread's activation depth, and the bold outlined threads span depth d. For any time step t during the execution and any activation depth d, let $s(t, d)$ denote the number of living threads that span d at the start of step t. Then the total space $s(t)$ being used at the start of time step t is

$$s(t) = \sum_{d=1}^{S_1} s(t, d) \ . \tag{5.1}$$

By induction on the number of steps, we shall show that for all t, every activation depth d has $s(t, d) \le P$. With this bound, equation (5.1) shows that $s(t) \le S_1 P$ for all time t, from which the space bound follows.

The algorithm begins with just one living thread (the root), so for every activation depth d, we have $s(1, d) \le 1 \le P$. Now, consider any activation depth d, and suppose that for time step t, the induction hypothesis $s(t, d) \le P$ holds. The computation being strict means that for each of the $s(t, d)$ living threads that span d at the start of step t, there is at least one ready thread with activation depth greater than or equal to d; remember, this is the crucial property that we get by having all data dependencies go from a child thread to an ancestor thread. Therefore, step t begins

with at least $s(t,d)$ ready threads at or deeper than d. The depth-first ordering then ensures that no more than $P - s(t,d)$ threads with depth less than d can execute at step t. Then, since the only way to increase the number of threads that span d is to execute a thread shallower than d that spawns a child thread at or deeper than d, step t ends with at most $s(t,d) + (P - s(t,d)) = P$ living threads that span activation depth d. Therefore, $s(t+1,d) \leq P$, and the induction is complete. □

Algorithm GDF′ is a refinement of Algorithm GDF that achieves greater efficiency by reducing the number of accesses to the global queue. Algorithm GDF′ begins with the root thread assigned to some arbitrary processor and the global queue empty. On subsequent steps, GDF′ has completed a "previous" step and must schedule threads for a "current" step. Suppose the previous step ends with P' out of the P processors not having a thread. To start the current step, the scheduler removes from the queue the P' deepest ready threads, or, if there are fewer than P' ready threads, it removes them all. It assigns these threads arbitrarily to the P' idle processors so that each idle processor receives at most one thread. The current step is now ready to proceed. Each of the P processors that has an assigned thread executes one task from that thread. Unless that thread spawns, dies, or stalls, the processor will have a thread at the end of the current step. If the thread stalls, then the processor must return it to the global queue, and consequently, the processor will not have a thread at the end of the current step. Similarly, if the thread dies, then the processor will not have a thread at the end of the step. Lastly, if the thread spawns a child, then the processor returns the parent thread (the one it was working on) to the global queue and keeps the child thread; in this case, the processor will still have a thread at the end of the current step.

Algorithm GDF′ achieves the same performance bounds as proved in Theorem 5.1, but it requires access to the global queue only when threads spawn, die, or stall.

THEOREM 5.2. *For any number P of processors and any strict multithreaded computation with work T_1, computation depth T_∞, and activation depth S_1, Algorithm* GDF′ *computes a P-processor schedule $\mathcal{X}$ that uses space $S(\mathcal{X}) \leq S_1 P$ and time $T(\mathcal{X}) \leq T_1/P + T_\infty$.*

Proof. This proof follows the proof of Theorem 5.1, but we add the following assertion to the induction hypothesis: for any activation depth d, if a step t begins with $s(t,d) \leq P$ living threads that span depth d, then step t begins with no more than $P - s(t,d)$ processors that have a thread with activation depth less than d. □

This algorithm may be feasible for a modest number of processors, but for a large number of processors, the cost of synchronization at the global queue becomes prohibitive. To derive a truly scalable and distributed algorithm, we need to split the global queue into P local queues, one for each processor. The next section presents and analyzes such a distributed algorithm.

We have been able to relate resource requirements to nonstrictness in the computation by characterizing two extremes. At one end, we have shown that arbitrary uses of nonstrictness make efficient execution impossible. At the other end, purely strict computations allow near optimally efficient executions. We now mention two minor results that begin to characterize resource requirements for limited uses of nonstrictness.

Given an arbitrary depth-first computation, any of the scheduling algorithms for strict computations can be employed by first adding data-dependency edges to make the computation strict. This transformation, known as *strictifying* (see Figure 5.1),

is always valid for depth-first computations. Of course, strictifying may dramatically reduce the average available parallelism, and therefore, we would like some way of exploiting the parallelism available through nonstrict spawns. Suppose we could execute the computation as if it were strictified, but at each step, if there is an idle processor and a thread that is stalled (due only to the strictness condition) at a task that wants to spawn, we let the processor go ahead and execute that task, thereby performing a nonstrict spawn. Unfortunately, the naive application of this rule can actually result in an execution that takes longer than the purely strict execution.

With due care, however, we can modify this rule to allow some nonstrict spawns while still guaranteeing the time and space bounds of a purely strict execution. For example, we can restrict the application of this rule to a set of threads designated by the programmer. If the programmer can designate this set of threads so as to ensure that, during execution, at most x nonstrictly spawned threads simultaneously span a given depth, then Algorithm GDF can achieve space bounded by $S_1(P + x)$ and linear speedup as in Theorem 5.1; similar results apply for Algorithm GDF$'$ and for the distributed algorithm that will be presented in the next section. Alternatively, by "sequestering" the nonstrictly spawned threads, the scheduler itself can budget the nonstrict spawns and achieve these same time and space bounds; details can be found in [4].

6. Distributed scheduling algorithms. In a distributed thread-scheduling algorithm, each processor works depth-first out of its own local priority queue. Specifically, to get a thread to work on, a processor removes the deepest ready thread from its local queue. Ideally, we would like the processor to then continue working on that thread until it either stalls, dies, or spawns, and when the processor does need to enqueue a thread (as in the case when the thread stalls or spawns) or dequeue a new thread, it does so by accessing only its local queue. Of course, this approach could result in processors with empty queues sitting idle while other processors have large queues. Thus, we require each processor to have some access to nonlocal queues in order to facilitate some type of load balancing.

The technique of Karp and Zhang [28] suggests a randomized algorithm in which threads are located in random queues in order to achieve some balance. We can show, however, that the naive adoption of this technique does not work. In particular, threads must migrate occasionally and some degree of synchronization is needed to avoid the large deviations that result if this random process is run over a long period of time. Further discourse on these problems can be found in [4]. In order to achieve the desired result, we modify the Karp and Zhang technique by incorporating a new mechanism to enforce a modest degree of synchrony among the processors.

Algorithm LDF (which stands for *local depth-first*) operates in iterations, with each iteration consisting of a synchronization phase followed by a computation phase and ending with a communication phase. In a synchronization phase, we compute a *cutoff depth* D which is a global value made available to all processors. During the following computation phase, only those threads with activation depth greater than or equal to D can execute. Finally, the communication phase redistributes threads to random locations.

The operation of each phase is governed by a *synchronization parameter* r that affects both the time and space performance of the algorithm. Let LDF(r) denote Algorithm LDF with synchronization parameter r.

In a synchronization phase of LDF(r), we use the synchronization parameter r to compute the cutoff depth D. Each processor p_i, for $i = 1, \ldots, P$, computes the

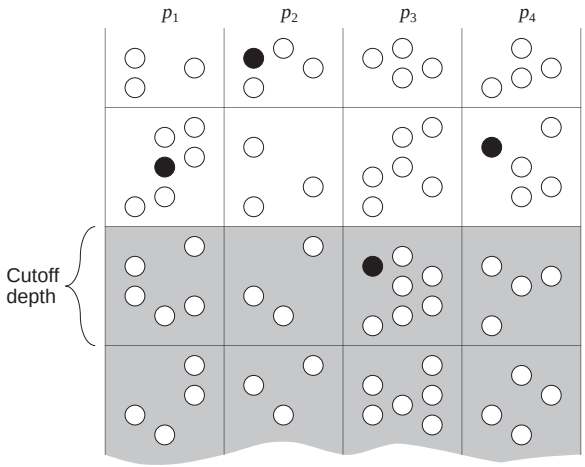


FIG. 6.1. *Computing the cutoff depth. Each column represents the local priority queue of a processor, and each row represents an activation depth with depth increasing in the downward direction. We depict each ready thread by a circle located in its processor's queue and at its activation depth. (Within a processor's queue, the horizontal ordering of threads is irrelevant.) The ready threads in each queue are ordered by activation depth with ties broken arbitrarily—this tie breaking is depicted by the vertical ordering of threads within an activation depth. In this example, the synchronization parameter* $r = 12$*, and the* 12*th deepest ready thread for each processor is shown in black (just count up from the bottom). The deepest of these black threads determines the cutoff depth. Only the ready threads at or deeper than the cutoff depth—those in the shaded region—can execute during the following computation phase.*

activation depth d_i of its rth deepest ready thread. In other words, d_i is the activation depth for which processor p_i has fewer than r ready threads deeper than d_i but at least r ready threads at or deeper than d_i. Cutoff depth D is then computed simply by

$$D = \max_{1 \le i \le P} d_i$$

as illustrated in Figure 6.1.

During the computation phase of LDF(r), each processor executes at least one task from each ready thread with activation depth greater than or equal to the cutoff depth D in its local queue. We further forbid each processor from executing more than r spawns; if a processor has more than r threads at or deeper than D that want to spawn, it may only execute r of them.

The iteration ends with a communication phase during which each processor must move each ready thread with activation depth greater than or equal to D (as determined at the beginning of the iteration) and each newly spawned thread from its local queue to a queue selected uniformly at random, independently for each thread.

By using the synchronization parameter r to compute the cutoff depth and then ensuring that each processor executes only tasks from threads at or deeper than the cutoff depth, while allowing at most r spawns, we get a guaranteed space bound.

LEMMA 6.1. *For any number* P *of processors and any strict multithreaded computation with activation depth* S_1*, Algorithm* LDF(r) *computes a* P*-processor schedule* $\mathcal{X}$ *such that* $S(\mathcal{X}) \le 2rS_1P$.

Proof. We show by induction on the number of iterations that no activation depth ever has more than $2rP$ living threads that span it. Specifically, recalling the

notation used in the proof of Theorem 5.1, we show that for every activation depth d and every iteration t of the execution, $s(t,d) \leq 2rP$. The result then follows from equation (5.1). As before, the base case is straightforward.

For any activation depth d and any iteration t of the execution, we consider two cases. In the first case, suppose iteration t begins with $rP \leq s(t,d) \leq 2rP$ living threads spanning depth d. Due to the strictness of the computation, there must be at least rP ready threads with activation depth greater than or equal to d, and by pigeonholing, some processor's local queue must have at least r of them. Therefore, the cutoff depth D will be set with $D \geq d$. Consequently, during the computation phase of iteration t, no thread with activation depth less than d can execute and the iteration ends with no more living threads spanning depth d than it started with. Now, suppose iteration t begins with $s(t,d) < rP$ living threads spanning depth d. In this case, during the computation phase, since each processor is only allowed r spawns, the number of living threads that span depth d can increase by at most rP, and therefore, the iteration ends with no more than $2rP$ living threads spanning depth d. In either case, $s(t+1,d) \leq 2rP$, which completes the induction. □

In order to achieve speedup in the execution time, we must ensure that during the computation phase of each iteration, each processor has some ready threads at or deeper than the cutoff depth. To ensure that the cutoff depth is not set too deep, we must use a large enough synchronization parameter r. On the other hand, the space bound of Lemma 6.1 is directly proportional to r. By setting $r = 6 \lg P$, the space bound of Lemma 6.1 becomes $S(\mathcal{X}) \leq 12 S_1 P \lg P$, and with high probability, most computation phases take $O(\lg P)$ time and get at least $P \lg P$ tasks executed as we now show.

To analyze the running time, we say that each iteration either *succeeds* or *fails* depending on how many tasks execute. An iteration that begins with at least $P \lg P$ ready threads fails if fewer than $P \lg P$ of the ready threads get a task executed. An iteration that begins with fewer than $P \lg P$ ready threads fails if not all of them get a task executed.

We now show that with the synchronization parameter set to $r = 6 \lg P$, it is highly likely that each iteration succeeds.

LEMMA 6.2. *For any number P of processors and any iteration of Algorithm* LDF$(6 \lg P)$*, the iteration fails with probability no more than P^{-5}.*

Proof. Suppose that when two threads have the same activation depth, we give each thread a unique identifier to break the tie so we can uniquely identify the $P \lg P$ deepest ready threads. If no local queue contains more than $6 \lg P$ of the $P \lg P$ deepest ready threads, then the synchronization phase sets the cutoff depth so that all $P \lg P$ of these deepest threads are at or are deeper than the cutoff depth. Therefore, an iteration succeeds if no local queue contains more than $6 \lg P$ of the $P \lg P$ deepest ready threads.

Consider a particular processor p_i, and let the random variable Z_i denote how many of the $P \lg P$ deepest ready threads start the iteration in the local queue of processor p_i. Each thread is located independently at random, and hence, the random variable Z_i has a binomial distribution with $P \lg P$ trials and success probability $1/P$. Therefore,

$$\Pr\{Z_i > 6 \lg P\} \leq \binom{P \lg P}{6 \lg P} \left(\frac{1}{P}\right)^{6 \lg P} .$$

Then, from the bound

$$\binom{x}{y} \leq \left(\frac{ex}{y}\right)^y \tag{6.1}$$

and the fact that $6 \geq 2e$, we can upper bound $\Pr\{Z_i > 6 \lg P\}$ by

$$\begin{aligned}\Pr\{Z_i > 6 \lg P\} &\leq \left(\frac{eP \lg P}{6 \lg P}\right)\left(\frac{1}{P}\right)^{6 \lg P} \\ &= \left(\frac{e}{6}\right)^{6 \lg P} \\ &\leq P^{-6}.\end{aligned}$$

Now, let $Z = \max_{1 \leq i \leq P} Z_i$. For an iteration that begins with at least $P \lg P$ ready threads, the probability of failure is no more than $\Pr\{Z > 6 \lg P\}$. We can use Boole's inequality to upper bound $\Pr\{Z > 6 \lg P\}$ by adding the individual probabilities, yielding

$$\begin{aligned}\Pr\{Z > 6 \lg P\} &\leq P \cdot \Pr\{Z_i > 6 \lg P\} \\ &\leq P^{-5} \ . \qquad \square\end{aligned}$$

We now show that iterations fail independently of each other. Specifically, we show that knowing whether an iteration t fails provides no information about whether any future iteration fails. The failure of an iteration depends only on how the ready threads are distributed among the processors. Therefore, we need to show that knowing whether iteration t fails provides no information about the distribution of threads at the end of the iteration. Suppose iteration t has cutoff depth D. No matter if iteration t fails or not, the iteration ends with a communication phase in which every ready thread at or deeper than D gets moved to a random location. Thus, iteration t provides no information about the distribution of threads at or deeper than the cutoff depth. Now, consider the threads less deep than D. The only part of an iteration that even considers the threads shallower than the cutoff depth is the synchronization phase. Therefore, we need to show that computing the cutoff depth provides no information about the distribution of threads with activation depth less than D. Consider an alternative method for computing the cutoff depth. Let all the processors work in synchrony from the bottom up. First each processor counts the number of ready threads it has with activation depth S_1. Then each processor adds on the number of ready threads it has with activation depth $S_1 - 1$. We continue in this manner until some processor reaches a count of r (the synchronization parameter). At this depth we stop and set the cutoff depth. In this way the synchronization phase can compute the cutoff depth with the exact same result but without ever considering threads shallower than D. Thus, computing the cutoff depth provides no information about the distribution of threads shallower than the cutoff depth.

With iterations failing independently of each other, we can bound the number of failed iterations, thereby bounding the total number of iterations taken.

LEMMA 6.3. *For any number P of processors and any strict multithreaded computation with work T_1 and computation depth T_∞, for any $\epsilon > 0$, with probability at least $1 - \epsilon$, Algorithm* LDF$(6 \lg P)$ *computes a P-processor schedule $\mathcal{X}$ that takes $O(T_1/(P \lg P) + T_\infty + \log_P(1/\epsilon))$ iterations.*

Proof. First we consider the failed iterations. Let the random variable f denote the number of failed iterations. We will show that for any $\epsilon > 0$, the probability that

$f \geq eT_1/(P \lg P) + b$ is no more than ϵ when $b = (1/3)\log_P(1/\epsilon)$. There are at most T_1 iterations, since each iteration always results in at least one task being executed, and each iteration fails independently with probability P^{-5}. Therefore, f is bounded by a binomial distribution with T_1 trials and success probability P^{-5}, from which we obtain

$$\Pr\left\{f \geq e\frac{T_1}{P\lg P} + b\right\} \leq \binom{T_1}{e\frac{T_1}{P\lg P} + b}\left(\frac{1}{P^5}\right)^{e\frac{T_1}{P\lg P}+b}.$$

Then, using inequality (6.1), we get

$$\begin{aligned}\Pr\left\{f \geq e\frac{T_1}{P\lg P} + b\right\} &\leq \left(\frac{eT_1}{e\frac{T_1}{P\lg P} + b}\cdot\frac{1}{P^5}\right)^{e\frac{T_1}{P\lg P}+b} \\ &\leq \left(\frac{P\lg P}{P^5}\right)^{e\frac{T_1}{P\lg P}+b} \\ &\leq \left(\frac{1}{P^3}\right)^{b} \\ &= P^{-3b},\end{aligned}$$

and $P^{-3b} = \epsilon$ for $b = (1/3)\log_P(1/\epsilon)$. Thus, with probability at least $1 - \epsilon$, we have $f = O(T_1/(P \lg P) + \log_P(1/\epsilon))$.

Now consider the successful iterations. We can think of each successful iteration as a step in a greedy schedule with $P \lg P$ processors. Then, as in the proof of the greedy-scheduling theorem (Theorem 3.1), we know that there can be no more than $T_1/(P \lg P) + T_\infty$ successful iterations.

Adding together the number of successful iterations and the number of failed iterations completes the proof. □

Now, if we let the random variable X_i denote the time taken by the ith computation phase of Algorithm LDF$(6 \lg P)$, we can give the total time in computation phases as the random variable $X = X_1 + X_2 + \cdots + X_Y$, where Y is the random variable denoting the number of iterations. The time taken by the ith computation phase is proportional to the maximum number of ready threads with activation depth greater than or equal to the cutoff depth in any processor. There can be a total of at most $18P \lg P$ ready threads at or deeper than the cutoff depth—$r = 6P \lg P$ deeper than the cutoff depth and $12P \lg P$ at the cutoff depth (from Lemma 6.1 with synchronization parameter $r = 6 \lg P$)—and each of these threads is located independently at random. Thus, we can bound each X_i as the size of the largest bin when throwing $18P \lg P$ balls at random into P bins. Furthermore, by the independence argument the X_i's are independent. We can now bound the random variable X.

LEMMA 6.4. *Let the random variable X denote the sum of Y mutually independent random variables, $X = X_1 + X_2 + \cdots + X_Y$ with each X_i, for $i = 1, \ldots, Y$, distributed as the number of balls in the fullest bin when throwing $P \ln P$ balls independently at random into $P \geq 2$ bins. Then, for any $\epsilon > 0$, we have $X = O(Y \ln P + \lg(1/\epsilon))$ with probability at least $1 - \epsilon$.*

Proof. We have

$$\begin{aligned}\Pr\{X \geq aY \ln P + b\} &= \Pr\left\{e^{X/e} \geq e^{(aY \ln P + b)/e}\right\} \\ &\leq \mathrm{E}\left[e^{X/e}\right] e^{-(aY \ln P + b)/e} \qquad (6.2)\end{aligned}$$

by Markov's inequality. By the independence of the X_i's,

$$\mathrm{E}\left[e^{X/e}\right] = \prod_{i=1}^{Y} \mathrm{E}\left[e^{X_i/e}\right]. \tag{6.3}$$

From the definition of expectation,

$$\mathrm{E}\left[e^{X_i/e}\right] = \sum_{j=\ln P}^{P\ln P} \Pr\{X_i = j\}\, e^{j/e}.$$

To bound $\mathrm{E}\left[e^{X_i/e}\right]$, we break this sum into pieces. First we break out the terms from $j = \ln P$ to $j = e^3 \ln P - 1$, which yields

$$\mathrm{E}\left[e^{X_i/e}\right] = \sum_{j=\ln P}^{e^3 \ln P - 1} \Pr\{X_i = j\}\, e^{j/e} + \sum_{j=e^3 \ln P}^{P\ln P} \Pr\{X_i = j\}\, e^{j/e}. \tag{6.4}$$

The first of these sums we bound by factoring out the largest term and upper-bounding the sum of probabilities by 1 as follows:

$$\begin{aligned}
\sum_{j=\ln P}^{e^3 \ln P - 1} \Pr\{X_i = j\}\, e^{j/e} &\le \sum_{j=\ln P}^{e^3 \ln P - 1} \Pr\{X_i = j\}\, e^{e^2 \ln P} \\
&= e^{e^2 \ln P} \sum_{j=\ln P}^{e^3 \ln P - 1} \Pr\{X_i = j\} \\
&\le e^{e^2 \ln P}. \qquad (6.5)
\end{aligned}$$

To bound the second sum in equation (6.4), we further break the range of the index variable j into smaller pieces indexed by $k = 3, \ldots, \lceil \ln P \rceil - 1$, with piece k going from $j = e^k \ln P$ to $j = e^{k+1} \ln P - 1$ as follows:

$$\begin{aligned}
\sum_{j=e^3 \ln P}^{P\ln P} \Pr\{X_i = j\}\, e^{j/e} &= \sum_{k=3}^{\lceil \ln P \rceil - 1} \left(\sum_{j=e^k \ln P}^{e^{k+1} \ln P - 1} \Pr\{X_i = j\}\, e^{j/e} \right) \\
&\le \sum_{k=3}^{\lceil \ln P \rceil - 1} \left(e^{e^k \ln P} \sum_{j=e^k \ln P}^{e^{k+1} \ln P - 1} \Pr\{X_i = j\} \right) \\
&\le \sum_{k=3}^{\lceil \ln P \rceil - 1} e^{e^k \ln P} \Pr\left\{X_i \ge e^k \ln P\right\} \\
&= \sum_{k=3}^{\lceil \ln P \rceil - 1} P^{e^k} \Pr\left\{X_i \ge e^k \ln P\right\}. \qquad (6.6)
\end{aligned}$$

Now we can bound $\Pr\left\{X_i \ge e^k \ln P\right\}$ by the same technique as in Lemma 6.2, since X_i has the same distribution as the random variable Z considered in the proof of Lemma 6.2:

$$\begin{aligned}
\Pr\left\{X_i \ge e^k \ln P\right\} &\le P \binom{P \ln P}{e^k \ln P} \left(\frac{1}{P}\right)^{e^k \ln P} \\
&\le P e^{-(k-1)e^k \ln P} \\
&= P^{-(k-1)e^k + 1}.
\end{aligned}$$

Substituting this bound into inequality (6.6) yields

$$\begin{aligned}\sum_{j=e^3 \ln P}^{P \ln P} \Pr\{X_i = j\}\, e^{j/e} &\le \sum_{k=3}^{\lceil \ln P \rceil - 1} P^{e^k} P^{-(k-1)e^k+1} \\ &\le \sum_{k=3}^{\infty} P^{-(k-2)e^k+1} \\ &\le 1, \qquad (6.7)\end{aligned}$$

since the sum is bounded by the geometric sum $\sum_{k=1}^{\infty} 2^{-k} = 1$. Now we can substitute inequalities (6.5) and (6.7) back into equation (6.4), producing

$$\begin{aligned}\mathrm{E}\left[e^{X_i/e}\right] &\le e^{e^2 \ln P} + 1 \\ &\le e^{(e^2+1)\ln P}.\end{aligned}$$

Finally, by substituting this bound into equation (6.3) and then substituting into inequality (6.2), we obtain

$$\begin{aligned}\Pr\{X \ge aY \ln P + b\} &\le e^{((e^2+1)\ln P)Y} e^{-(aY \ln P + b)/e} \\ &= \exp\left(-\left(\frac{a}{e} - e^2 - 1\right) Y \ln P - \frac{b}{e}\right) \\ &\le \exp\left(-\frac{b}{e}\right)\end{aligned}$$

for $a \ge e^3 + e$. Thus, with $b = e\ln(1/\epsilon)$, we obtain

$$\Pr\left\{X \ge (e^3 + e)Y \ln P + e\ln(1/\epsilon)\right\} \le \epsilon. \qquad \square$$

We can now characterize the time and space usage for execution schedules computed by the LDF algorithm with synchronization parameter $r = 6 \lg P$.

THEOREM 6.5. *For any number $P \ge 2$ of processors and any strict multithreaded computation with work T_1, computation depth T_∞, and activation depth S_1, Algorithm* LDF$(6 \lg P)$ *computes a P-processor schedule $\mathcal{X}$ that uses space $S(\mathcal{X}) = O(S_1 P \lg P)$, and for any $\epsilon > 0$, with probability at least $1 - \epsilon$, the schedule uses time $T(\mathcal{X}) = O(T_1/P + T_\infty \lg P + \lg(1/\epsilon))$.*

Proof. The space bound follows directly from Lemma 6.1 with synchronization parameter $r = 6 \lg P$. The time $T(\mathcal{X})$ is the total time taken in computation phases. Let the random variable Y denote the number of iterations. Then we can decompose $T(\mathcal{X})$ as a sum of Y mutually independent random variables, $T(\mathcal{X}) = X_1 + X_2 + \cdots + X_Y$, with each X_i distributed as the size of the fullest bin when throwing $18P \lg P$ balls independently at random into P bins. Using $\epsilon/2$ as the value of ϵ in Lemma 6.3, we obtain $Y = O(T_1/(P \lg P) + T_\infty + \log_P(1/\epsilon))$ with probability at least $1 - \epsilon/2$. Then, using $\epsilon/2$ as the value of ϵ in Lemma 6.4, we obtain $T(\mathcal{X}) = O(Y \lg P + \lg(1/\epsilon))$ with probability at least $1 - \epsilon/2$ (using $18P \lg P$ instead of $P \ln P$ only affects the constant). Thus, with probability at least $1 - \epsilon$, the total time taken in computation phases is $T(\mathcal{X}) = O(T_1/P + T_\infty \lg P + \lg(1/\epsilon))$. $\square$

COROLLARY 6.6. *For any number $P \ge 2$ of processors and any strict multithreaded computation with work T_1 and computation depth T_∞, Algorithm* LDF$(6 \lg P)$

computes a P-processor schedule $\mathcal{X}$ with expected execution time $\mathrm{E}[T(\mathcal{X})] = O(T_1/P + T_\infty \lg P)$.

Proof. Just use $\epsilon = 1/P$ in Theorem 6.5 to get $T(\mathcal{X}) = O(T_1/P + T_\infty \lg P)$ with probability at least $1 - 1/P$. Then we have

$$\begin{aligned}\mathrm{E}[T(\mathcal{X})] &\le \left(1 - \frac{1}{P}\right) O\left(\frac{T_1}{P} + T_\infty \lg P\right) + \frac{1}{P} T_1 \\ &= O\left(\frac{T_1}{P} + T_\infty \lg P\right). \qquad \square\end{aligned}$$

This algorithm achieves linear expected speedup when the computation has average available parallelism $T_1/T_\infty = \Omega(P \lg P)$.

We can view the $\lg P$ factors in the space bound and the average available parallelism required to achieve linear speedup as the computational slack required by Valiant's bulk-synchronous model [42]. The space bound $S(\mathcal{X}) = O(S_1 P \lg P)$ indicates that Algorithm LDF$(6 \lg P)$ requires memory to scale sufficiently to allow each *physical* processor enough space to simulate $\Theta(\lg P)$ *virtual* processors. Given this much space, the time bound $\mathrm{E}[T(\mathcal{X})] = O(T_1/P + T_\infty \lg P)$ then demonstrates linear expected speedup provided the computation has $\lg P$ slack in the average available parallelism.

The space bound of Theorem 6.5 is an aggregate bound, but in a distributed memory machine, we may want to bound the space associated with each individual processor's queue. In the LDF algorithm, each living thread is located in the local queue of a processor chosen at random, so we assume that each activation frame is located in the local memory of the same randomly chosen processor as its associated living thread. Since the aggregate space used by Algorithm LDF(r) is bounded by $2rS_1P$, we would like some way to ensure that each individual processor requires space bounded by $O(rS_1)$.

If we consider any given processor p and any given iteration t of the algorithm, then we can let W denote the total space being used by activation frames located in the memory of processor p. We can decompose W as a weighted sum of independent indicator random variables and show that $\mathrm{E}[W] \le 2rS_1$. Then, using a theorem due to Raghavan [36, Theorem 1], we can show that with probability at least $1 - e^{-2r}$, we have $W \le 2erS_1$.

With this probabilistic bound on the space used by a given processor at a given iteration, we can show that with appropriate choice of the synchronization parameter r, we can bound the per-processor memory by simply rerandomizing thread locations any time a processor's memory fills up. In particular, if we choose $r = \Theta(\lg P + \lg S_1)$, then the total time spent rerandomizing is $O(T_1/P)$ and the per-processor storage bound is $O(S_1(\lg P + \lg S_1))$. Details can be found in [4].

7. Related and future work. Although the work we have presented here provides some theoretical underpinnings for understanding the resource requirements of multithreaded computations, much remains to be done. In this section, we review some of the related work, both theoretical and empirical, on scheduling dynamic computations. We discuss the class of "thread-stealing" algorithms and present some of our preliminary research on this kind of scheduling algorithm.

Substantial research has been reported in the theoretical literature concerning dynamic computations. In contrast to our research on multithreaded computations, however, other theoretical research has tended to treat the aggregate resource requirements of a computation as a given, rather than as a quantity that depends on the

execution schedule. Thus, the relevant issue in this work is how to balance the load across processors. Important work in this area includes a randomized work-stealing algorithm for load balancing [38]; dynamic tree-embedding algorithms [3, 31]; and algorithms for backtrack search [27, 37, 45], which can be viewed as a multithreaded computation with no data-dependency edges. Although this work ignores aggregate space requirements, it is interesting to note that Zhang's work-stealing algorithm for backtrack search [45] actually gives at most linear expansion of space, but he does not mention this fact.

The problem of storage management for multithreaded computations has been a growing concern among practitioners [13, 22]. To date, most existing techniques for controlling storage requirements have consisted of heuristics to either bound storage use by explicitly controlling storage as a resource or reduce storage use by modifying the scheduler's behavior. We are aware of no prior scheduling algorithms for multithreaded computations for which simultaneously good time and space bounds have been proved.

The storage management problem can often be quite pronounced under the execution of a *fair* scheduler. By executing threads in round-robin fashion, a fair scheduler gives each ready thread a fair portion of the execution time. A fair scheduler aggressively exposes parallelism, often resulting in excessive space requirements. In order to curb the excessive use of space exhibited by fair scheduling, researchers from the dataflow community have developed heuristics to explicitly manage storage [15, 39]. The effectiveness of these heuristics is documented with encouraging empirical evidence but no provable time bounds.

In contrast with these heuristic techniques, we have chosen to develop an algorithmic foundation that manages storage by allowing programmers to leverage their knowledge of storage requirements for serially executed programs.

Other researchers have also addressed the storage issue by attempting to relate parallel storage requirements to serial storage requirements. Burton and Sleep [12] and Halstead [22], for example, considered unfair scheduling policies based on *thread stealing*. In these thread-stealing strategies, each processor works depth-first—just like a serial execution—but when a processor runs out of ready threads, it *steals* threads from other processors. In many cases, this scheduling policy results in each processor using no more space than that used by a single processor, but a problem arises as to what to do when all threads in a processor have stalled. If the processor goes out to steal a thread from another processor, greater-than-linear space expansion may result. If the processor goes idle, however, linear speedup is not guaranteed. For these unfair scheduling policies, characterizing the performance analytically is difficult.

Thread stealing has also been employed in two parallel chess-playing programs. Zugzwang [18] is a program in which processors steal subcomputations of a chess tree using a parallel alpha-beta search algorithm. StarTech [30] is another parallel program organized along similar lines but with a parallel scout-search algorithm. Although the authors make no guarantees of performance for their algorithms, the empirical results of these programs are good; both have won prizes in international chess competitions.

In recent work, we have obtained some preliminary results on thread stealing. We have devised a new global algorithm that forms the basis of a randomized, distributed, thread-stealing algorithm. Our new global algorithm is like GDF$'$ except for two changes. First, the global queue is not organized by activation depth; when a processor removes a ready thread from the queue, any ready thread suffices. Second, when a thread dies, the thread's processor must locate the parent thread in the global queue

and check to see if the parent has any surviving children. If the parent no longer has any surviving children, then the processor must commence work on the parent thread. Otherwise, the processor is free to take any ready thread from the global queue. It can be proved by simple induction that this algorithm satisfies the same time and space bounds as Algorithms GDF and GDF′ . In our distributed thread-stealing algorithm, we replace the global queue with local queues, one per processor. By making some generous modeling assumptions, we have been able to analyze this algorithm and to obtain bounds similar to those for Algorithm LDF. We are currently working on improving these results.

Appendix. During the time between our results becoming publicly known [7] and this journal publication, we have explored multithreaded computing more fully. We have been able to characterize the performance of a distributed thread-stealing algorithm [5, 8]. For the class of "fully strict" (well-structured) computations, this randomized algorithm achieves execution space bounded by S_1P and expected execution time bounded by $O(T_1/P+T_\infty)$, including scheduling overheads. Additionally, in contrast to Algorithm LDF, this thread-stealing algorithm is efficient with respect to communication. We have implemented this thread-stealing algorithm in the runtime system for *Cilk* [5, 6], a parallel multithreaded extension of the C language. By employing a provably efficient scheduler, Cilk is able to deliver efficient and predictable performance, guaranteed. Moreover, structure in the Cilk programming model facilitates the implementation of "adaptive parallelism" and transparent fault tolerance in a runtime system for Cilk on networks of workstations [5, 9]. More information about Cilk is available online at `http://theory.lcs.mit.edu/~cilk`.

Acknowledgments. The authors thank Bonnie Berger, Tom Cormen, Esther Jesurum, Mike Klugerman, Bradley Kuszmaul, Tom Leighton, Arthur Lent, Greg Papadopoulos, Atul Shrivastava, and Ethan Wolf of the MIT Laboratory for Computer Science for insightful discussions.

REFERENCES

[1] A. Agarwal, B.-H. Lim, D. Kranz, and J. Kubiatowicz, *APRIL: A processor architecture for multiprocessing*, in Proc. 17th Annual Intl. Symposium on Computer Architecture, Seattle, WA, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 104–114; Tech. Report MIT/LCS/TM-450, MIT Laboratory for Computer Science, Cambridge, MA, 1991.

[2] W. C. Athas and C. L. Seitz, *Multicomputers: Message-passing concurrent computers*, Computer, 21 (1988), pp. 9–24.

[3] S. Bhatt, D. Greenberg, T. Leighton, and P. Liu, *Tight bounds for on-line tree embeddings*, in Proc. of the Second Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, SIAM, Philadelphia, 1991, pp. 344–350.

[4] R. D. Blumofe, *Managing Storage for Multithreaded Computations*, Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1992; Tech. Report MIT/LCS/TR-552, MIT Laboratory for Computer Science, Cambridge, MA, 1992.

[5] R. D. Blumofe, *Executing Multithreaded Programs Efficiently*, Ph.D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1995.

[6] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, *Cilk: An efficient multithreaded runtime system*, in Proc. of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), Santa Barbara, CA, ACM, New York, 1995, pp. 207–216.

[7] R. D. Blumofe and C. E. Leiserson, *Space-efficient scheduling of multithreaded computations*, in Proc. of the 25th Annual ACM Symposium on Theory of Computing (STOC), San Diego, CA, ACM, New York, 1993, pp. 362–371.

[8] R. D. Blumofe and C. E. Leiserson, *Scheduling multithreaded computations by work stealing*, in Proc. of the 35th Annual Symposium on Foundations of Computer Science (FOCS), Santa Fe, NM, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 356–368.

[9] R. D. Blumofe and D. S. Park, *Scheduling large-scale parallel computations on networks of workstations*, in Proc. of the Third Intl. Symposium on High Performance Distributed Computing (HPDC), San Francisco, CA, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 96–105.

[10] B. Boothe and A. Ranade, *Improved multithreading techniques for hiding communication latency in multiprocessors*, in Proc. of the 19th Annual Intl. Symposium on Computer Architecture, Gold Coast, Australia, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 214–223.

[11] R. P. Brent, *The parallel evaluation of general arithmetic expressions*, J. Assoc. Comput. Mach., 21 (1974), pp. 201–206.

[12] F. W. Burton and M. R. Sleep, *Executing functional programs on a virtual tree of processors*, in Proc. of the 1981 Conference on Functional Programming Languages and Computer Architecture, Portsmouth, NH, ACM, New York, 1981, pp. 187–194.

[13] D. E. Culler, *Resource Management for the Tagged Token Dataflow Architecture*, Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1980; Technical Report MIT/LCS/TR-332, MIT Laboratory for Computer Science, Cambridge, MA, 1985.

[14] D. E. Culler, *Managing Parallelism and Resources in Scientific Dataflow Programs*, Ph.D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1990; Tech. Report MIT/LCS/TR-446, MIT Laboratory for Computer Science, Cambridge, MA, 1990.

[15] D. E. Culler and Arvind, *Resource requirements of dataflow programs*, in Proc. of the 15th Annual Intl. Symposium on Computer Architecture, Honolulu, HI, IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 141–150; Computation Structures Group Memo 280, MIT Laboratory for Computer Science, Cambridge, MA, 1987.

[16] D. E. Culler, A. Sah, K. E. Schauser, T. von Eicken, and J. Wawrzynek, *Fine-grain parallelism with minimal hardware support: A compiler-controlled threaded abstract machine*, in Proc. of the Fourth Intl. Conference on Architectural Support for Programming Languages and Operating Systems, Santa Clara, CA, ACM, New York, 1991, pp. 164–175.

[17] W. J. Dally, L. Chao, A. Chien, S. Hassoun, W. Horwat, J. Kaplan, P. Song, B. Totty, and S. Wills, *Architecture of a message-driven processor*, in Proc. of the 14th Annual Intl. Symposium on Computer Architecture, Pittsburgh, PA, IEEE Computer Society Press, Los Alamitos, CA, 1987, pp. 189–196.

[18] R. Feldmann, P. Mysliwietz, and B. Monien, *Game tree search on a massively parallel system*, Adv. Comput. Chess, 7 (1993), pp. 203–219.

[19] V. G. Grafe and J. E. Hoch, *The Epsilon-2 hybrid dataflow architecture*, in Proc. 35th IEEE Computer Society Intl. Computer Conf. (COMPCON 90), San Francisco, CA, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 88–93.

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

[21] R. L. Graham, *Bounds on multiprocessing timing anomalies*, SIAM J. Appl. Math., 17 (1969), pp. 416–429.

[22] R. H. Halstead, Jr., *Multilisp: A language for concurrent symbolic computation*, ACM Trans. Prog. Lang. Syst., 7 (1985), pp. 501–538.

[23] R. H. Halstead, Jr. and T. Fujita, *MASA: A multithreaded processor architecture for parallel symbolic computing*, in Proc. of the 15th Annual Intl. Symposium on Computer Architecture, Honolulu, HI, IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 443–451.

[24] W. Horwat, *Concurrent Smalltalk on the Message-Driven Processor*, Tech. Report MIT/AI/TR-1321, MIT Artificial Intelligence Laboratory, Cambridge, MA, 1991.

[25] R. A. Iannucci, *Toward a dataflow/von Neumann hybrid architecture*, in Proc. of the 15th Annual Intl. Symposium on Computer Architecture, Honolulu, HI, IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 131–140; Computation Structures Group Memo 275, MIT Laboratory for Computer Science, Cambridge, MA, 1988.

[26] S. Jagannathan and J. Philbin, *A customizable substrate for concurrent languages*, in Proc. of the ACM SIGPLAN '92 Conference on Programming Language Design and Implementation, San Francisco, CA, ACM, New York, 1992, pp. 55–67.

[27] C. Kaklamanis and G. Persiano, *Branch-and-bound and backtrack search on mesh-connected arrays of processors*, in Proc. of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures, San Diego, CA, ACM, New York, 1992, pp. 118–126.

[28] R. M. Karp and Y. Zhang, *A randomized parallel branch-and-bound procedure*, in Proc. of the 20th Annual ACM Symposium on Theory of Computing, Chicago, IL, ACM, New York, 1988, pp. 290–300.

[29] S. W. Keckler and W. J. Dally, *Processor coupling: Integrating compile time and runtime scheduling for parallelism*, in Proc. of the 19th Annual Intl. Symposium on Computer Architecture, Gold Coast, Australia, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 202–213.

[30] B. C. Kuszmaul, *Synchronized MIMD Computing*, Ph.D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1994; Tech. Report MIT/LCS/TR-645, MIT Laboratory for Computer Science, 1994; also available online via ftp://theory.lcs.mit.edu/pub/bradley/phd.ps.Z.

[31] T. Leighton, M. Newman, A. G. Ranade, and E. Schwabe, *Dynamic tree embeddings in butterflies and hypercubes*, in Proc. of the 1989 ACM Symposium on Parallel Algorithms and Architectures, Santa Fe, NM, ACM, New York, 1989, pp. 224–234.

[32] E. Mohr, D. A. Kranz, and R. H. Halstead, Jr., *Lazy task creation: A technique for increasing the granularity of parallel programs*, IEEE Trans. Parallel Distrib. Systems, 2 (1991), pp. 264–280.

[33] R. S. Nikhil and Arvind, *Can dataflow subsume von Neumann computing?*, in Proc. of the 16th Annual Intl. Symposium on Computer Architecture, Jerusalem, Israel, IEEE Computer Society Press, Los Alamitos, CA, 1989, pp. 262–272; Computation Structures Group Memo 292, MIT Laboratory for Computer Science, Cambridge, MA, 1989.

[34] R. S. Nikhil, G. M. Papadopoulos, and Arvind, *$^*T$: A multithreaded massively parallel architecture*, in Proc. of the 19th Annual Intl. Symposium on Computer Architecture, Gold Coast, Australia, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 156–167; Computation Structures Group Memo 325–1, MIT Laboratory for Computer Science, Cambridge, MA, 1991.

[35] G. M. Papadopoulos and K. R. Traub, *Multithreading: A revisionist view of dataflow architectures*, in Proc. of the 18th Annual Intl. Symposium on Computer Architecture, Toronto, Canada, IEEE Computer Society Press, Los Alamitos, CA, 1991, pp. 342–351; Computation Structures Group Memo 330, MIT Laboratory for Computer Science, Cambridge, MA, 1991.

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

[37] A. Ranade, *Optimal speedup for backtrack search on a butterfly network*, in Proc. of the Third Annual ACM Symposium on Parallel Algorithms and Architectures, Hilton Head, SC, ACM, New York, 1991, pp. 40–48.

[38] L. Rudolph, M. Slivkin-Allalouf, and E. Upfal, *A simple load balancing scheme for task allocation in parallel machines*, in Proc. of the Third Annual ACM Symposium on Parallel Algorithms and Architectures, Hilton Head, SC, ACM, New York, 1991, pp. 237–245.

[39] C. A. Ruggiero and J. Sargeant, *Control of parallelism in the Manchester dataflow machine*, in Functional Programming Languages and Computer Architecture, Lecture Notes in Comput. Sci., 274, Springer-Verlag, Berlin, 1987, pp. 1–15.

[40] M. Sato, Y. Kodama, S. Sakai, Y. Yamaguchi, and Y. Koumura, *Thread-based programming for the EM-4 hybrid dataflow machine*, in Proc. of the 19th Annual Intl. Symposium on Computer Architecture, Gold Coast, Australia, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 146–155.

[41] K. R. Traub, *Sequential Implementation of Lenient Programming Languages*, Ph.D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1988; Tech. Report MIT/LCS/TR-417, MIT Laboratory for Computer Science, Cambridge, MA, 1988.

[42] L. G. Valiant, *A bridging model for parallel computation*, Comm. ACM, 33 (1990), pp. 103–111.

[43] M. T. Vandevoorde and E. S. Roberts, *WorkCrews: An abstraction for controlling parallelism*, Internat. J. Parallel Programming, 17 (1988), pp. 347–366.

[44] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser, *Active messages: A mechanism for integrated communication and computation*, in Proc. of the 19th Annual Intl. Symposium on Computer Architecture, Gold Coast, Australia, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 256–266.

[45] Y. Zhang, *Parallel Algorithms for Combinatorial Search Problems*, Ph.D. thesis, Department of Electrical Engineering and Computer Science, University of California at Berkeley, 1989; Tech. Report UCB/CSD 89/543, University of California at Berkeley, Computer Science Division, 1989.

SIAM J. COMPUT.
Vol. 27, No. 1, pp. 230–246, February 1998
© 1998 Society for Industrial and Applied Mathematics
010

SIMULATING THRESHOLD CIRCUITS BY MAJORITY CIRCUITS*

MIKAEL GOLDMANN† AND MAREK KARPINSKI‡

Abstract. We prove that a single threshold gate with arbitrary weights can be simulated by an explicit polynomial-size, depth-2 majority circuit. In general we show that a polynomial-size, depth-d threshold circuit can be simulated uniformly by a polynomial-size majority circuit of depth $d+1$. Goldmann, Håstad, and Razborov showed in [*Comput. Complexity*, 2 (1992), pp. 277–300] that a nonuniform simulation exists. Our construction answers two open questions posed by them: we give an explicit construction, whereas they use a randomized existence argument, and we show that such a simulation is possible even if the depth d grows with the number of variables n (their simulation gives polynomial-size circuits only when d is constant).

Key words. threshold circuits, majority circuits, circuit complexity

AMS subject classifications. 68Q05, 68Q22, 68Q25

PII. S0097539794274519

1. Introduction.

A threshold gate is a fairly simple device that computes a weighted sum of its inputs, compares it to a threshold value, and outputs 1 or 0 depending on the outcome of the comparison. A threshold circuit is an acyclic network of threshold gates. The *size* of a circuit is the number of wires in it.

Small-weight threshold gates are a restricted type of threshold gates. In this case the magnitude of the (integer) weights of the gate is bounded by a polynomial in the number of inputs to the gate. The corresponding circuits are called small-weight threshold circuits. In this case the magnitude of the weights in the circuit is bounded by a polynomial in the total number of inputs to the circuit. It is easy to see that a majority gate can simulate a small weight threshold gate by simply duplicating input wires and adding some constant inputs. This leads only to a polynomial increase in the number of wires. Hence, depth-d, polynomial-size majority circuits are equivalent to depth d polynomial-size, small-weight threshold circuits.

Threshold circuits have been shown to be surprisingly powerful. It is implicit in work by Beame, Cook, and Hoover [4] that integer division can be carried out by polynomial-size threshold circuits of constant depth. Allender [1] (inspired by Toda [29]) shows that any function in AC^0 can be computed by depth-3 majority circuits of quasi-polynomial size. Yao [34] extends this to all of ACC^0 (see also [5]).

There are some strong lower bounds for majority circuits of very small depth. Hajnal et al. [11] prove exponential lower bounds on the size of depth-2 majority circuits computing "inner product mod 2." These results were extended in [13] to depth-3 majority circuits where the gates on the bottom level have very small fan-in, and recently superpolynomial bounds were proved for depth-3 majority circuits where the gates on the bottom level are arbitrary gates of fan-in $n^{1-\epsilon}$ [23]. For depth-3 majority circuits with no extra restrictions, no superpolynomial lower bounds are

* Received by the editors September 12, 1994; accepted for publication (in revised form) January 18, 1996. A preliminary version of this paper appeared in *Proc. 25th Annual ACM Symposium on the Theory of Computing*, ACM, New York, 1993, pp. 551–560.

http://www.siam.org/journals/sicomp/27-1/27451.html

† Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, Sweden (migo@nada.kth.se). This author's work was done in part while visiting the University of Bonn.

‡ Department of Computer Science, University of Bonn, 53117 Bonn (marek@cs.bonn.edu) and International Computer Science Institute, Berkeley, CA 94704. This author was supported in part by Leibniz Center for Research in Computer Science, by DFG grant KA 673/4-1, and by ESPRIT BR grant 7097.

known. Siu, Roychowdhury, and Kailath proved superlinear lower bounds on the number of wires in constant-depth majority circuits computing parity [28]. Recently Impagliazzo, Paturi, and Saks extended the results of [28] to constant-depth threshold circuits (with large weights) computing parity [15].

If one considers threshold circuits with arbitrary weights, even less is known. There is no superpolynomial lower bound for depth-2 threshold circuits computing some function in NP. As we mentioned above, such bounds exist for majority circuits [11]. It is therefore interesting to explore the power of large weights in threshold circuits.

It has long been known that a single threshold gate is strictly more powerful than a single small-weight threshold gate. For instance, Myhill and Kautz showed in 1961 that there are functions computable by a single threshold gate that require weights of size $\Omega(2^n/n)$, where n is the number of inputs [17]. Also, it was recently shown that depth-2, polynomial-size threshold circuits are more powerful than depth-2, polynomial-size, small-weight threshold circuits [10].

On the other hand, it was proved by Chandra, Stockmeyer, and Vishkin [8] that addition of n binary encoded integers can be performed by constant depth majority circuits (see also [21]). This implies that depth-d, polynomial-size threshold circuits can be simulated by depth $O(d)$, polynomial-size, small-weight threshold circuits.

In [25] Siu and Bruck gave a nonconstructive proof that polynomial-size, depth-d threshold circuits can be simulated by polynomial-size, depth-$2d+1$ majority circuits. Alon and Bruck gave a uniform construction in [3] achieving this. Goldmann, Håstad, and Razborov showed in [10] that any function computed by a depth-d, polynomial-size threshold circuit is computable by a depth-$d+1$, polynomial-size majority circuit (note that for $d = 1, 2$ this is optimal). Two open questions were posed in [10]: can one make an explicit construction, and can one make it work also for nonconstant depth? (Reference [10] uses a probabilistic argument, and the blowup in size is superpolynomial if the depth grows with the number of variables.) We give positive answers to both questions.

For a thorough survey of complexity theoretic results on threshold circuits, see [22].

2. Preliminaries. It will be convenient to work over $\{1, -1\}$ rather than $\{0, 1\}$. An n-variable Boolean function thus maps $\{1, -1\}^n$ to $\{1, -1\}$. This is a simple transformation that does not affect the power of threshold gates. A threshold gate computes its output as the sign of a linear form:

$$g(x) = \operatorname{sign}\left(w_0 + \sum_{i=1}^{n} w_i x_i\right),$$

where the w_i are the *weights*, and the sign function takes a real-valued input and is defined by

$$\operatorname{sign}(x) = \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$

Since g is to be ± 1-valued, the argument to the sign function is required to be nonzero for all $x \in \{1, -1\}^n$. The following well-known result by Muroga [16] gives a bound on the magnitude of the weights.

THEOREM 2.1 (see [16, Theorem 9.3.2.1]). *Let $f(x)$ be an arbitrary n-variable threshold function. Then $f(x)$ can be written as*

$$f(x) = \operatorname{sign}\left(w_0 + \sum_1^n w_i x_i\right),$$

where for all $i = 0, \ldots, n$,

$$w_i \in \mathbb{Z} \quad \textit{and} \quad |w_i| \leq 2^{-n}(n+1)^{(n+1)/2}.$$

Remark 1. A recent result by Håstad [12] gives a lower bound on the weights required for a particular threshold function. The lower bound nearly matches the upper bound in Theorem 2.1.

A threshold circuit is a directed acyclic graph the nodes of which are either threshold gates or input variables. We allow multiple output nodes, and a circuit is evaluated in topological order. For a given circuit C, f_C is the function computed by C. The size of a circuit is the number of wires in it.

A family of circuits $\{C_n\}$ is said to compute a function $f\colon \{1,-1\}^* \to \{1,-1\}^*$ provided $f \restriction_{\{1,-1\}^n} = f_{C_n}$ for all n.

DEFINITION 2.2. *We use the following notation from* [6].

- *LT_d is the class of functions computable by depth-d, polynomial-size threshold circuits.*
- *$\widehat{LT}_d$ is the class of functions computable by depth-d, polynomial-size threshold circuits with polynomially bounded integer weights.*

Note that if we allow a gate to have multiple wires from an input (i.e., the underlying structure is a *multi*-graph), then depth-d, polynomial-size majority circuits can compute any function in $\widehat{LT}_d$.

We introduce also uniform classes of threshold circuits. In doing so we can use the threshold circuit descriptions, e.g., similar to the "direct connection language" of Ruzzo [24] (see also [9]).

Let BIN be the set of binary encodings of the positive integers. A threshold circuit C_n with n inputs is described by a labeled multigraph (V_n, E_n). $V_n = I_n \dot\cup G_n$, where $I_n = \{1, x_1, \ldots, x_n\}$ is the set of inputs (note that the constant "1" is included in I_n), and $G_n = \{g_1, \ldots, g_s\}$ is the set of gate labels. The edges in E_n are tuples $(e, v, g, k, \delta) \in V_n \times BIN \times G_n \times BIN \times \{1, -1\}$ with the following interpretation: the edge with label e goes from v to g (where e is a unique edge label), and the weight of the edge is $\delta \cdot 2^k$. There is no need to assign a threshold value to the gates in G_n. We give them all threshold 0 and by having an appropriate weight assigned to the constant input 1 any threshold can be achieved. Also, an edge with weight w can be split into about $\log w$ edges the weights of which are powers of 2. We use $\overline{C_n}$ to denote the description of C_n.

We use the same syntax when describing small-weight threshold circuits, but k in an edge-tuple (n, e, v, g, k, δ) is interpreted as the weight having magnitude k rather than 2^k.

A circuit-family has *polynomial-size descriptions* if there is a constant c such that

- for each $n \in \mathbb{N}$ and each $g \in G_n$ it holds that $|g| \leq c(1 + \log(n+1))$.
- for each $n \in \mathbb{N}$ and each edge $(e, v, g, k, \delta) \in E_n$ it holds that $|e| \leq c(1 + \log(n+1))$ and that $|k| \leq c(1 + \log(n+1))$.

Note that given the above constraints both V_n and E_n have size polynomial in n.

To define uniformity, let $\mathcal{C} = \{C_n\}$ be a sequence of circuits, and for each C_n let G_n and E_n be the corresponding gates and edges. $\mathcal{C}$ defines the language $L_{\mathcal{C}} = L_G \cup L_E$, where $L_G = \{ (n, g) \mid g \in G_n \}$ and $L_E = \{ (n, e, v, g, k, \delta) \mid (e, v, g, k, \delta) \in E_n \}$.

DEFINITION 2.3. *A function is in L-uniform LT_d if there is a sequence $\mathcal{C}$ of depth-d threshold circuits with polynomial-size descriptions, such that there is a Turing machine that accepts $L_{\mathcal{C}}$ in linear space.*

A function is in L-uniform $\widehat{LT}_d$ if there is a sequence $\mathcal{C}$ of depth-d small-weight threshold circuits with polynomial-size descriptions, such that there is Turing machine that accepts $L_{\mathcal{C}}$ in linear space.

The reason for the terminology "L-uniform" is that the description of a circuit of the above type can be generated in space that is logarithmic in the size of the description.

We call a sequence of circuits P-uniform if the description of the nth circuit can be generated in polynomial time on input 1^n (cf., for example, [9]).

We define the following operator.

DEFINITION 2.4. *The operator* rem *is defined as follows. For integers a and b,*

$$a \text{ rem } b = c,$$

where c is the unique integer such that

$$a \equiv c \pmod{b} \quad \textit{and} \quad -b/2 < c \leq b/2.$$

3. The idea behind the construction. Let f be an arbitrary threshold gate given by

$$f(x) = \operatorname{sign}(F(x)),$$

where

$$F(x) = w_0 + \sum_{i=1}^{n} w_i x_i.$$

Since we have integer weights, and we require that the argument of 'sign' is nonzero, we have $|F(x)| \geq 1$ for all x.

In the next section we will build a parametrized *approximator* for f. For a fixed input, the approximator is good for randomly chosen parameters. To show the intuition behind the construction, we build an approximator φ for f that computes correctly for a random input.

In the construction it will be convenient to use rational weights. All the weights in the circuits we construct are of the form $w/2$, where w is an integer. By multiplying all weights in the circuit by 2 we get integer weights, and the increase in the magnitude of the weights is just a factor 2.

To describe the construction we need a couple of parameters that we call W and m. Let $w_{\max}(f)$ be the largest magnitude of a weight of the gate f. It will be convenient to assume the following:

$$W \geq \max\{w_{\max}(f), 2^n\}, \tag{1}$$

$$10n \leq m \leq W. \tag{2}$$

The parameter m controls the quality of the approximator. In the circuit that implements the approximator the weights will have magnitude about m^2. To have small-weight circuits, clearly m must be kept small. On the other hand, intuitively it seems that allowing larger weights allows a more accurate approximator. The parameter m reflects this trade-off.

The construction uses the following function of one integer variable y:

$$\begin{aligned} M^m(y) = &\frac{1}{2}\operatorname{sign}\left(y - m + \frac{1}{2}\right) \\ &-\frac{1}{2}\operatorname{sign}\left(y - 2m + \frac{1}{2}\right) \\ &+\frac{1}{2}\operatorname{sign}\left(y + m - \frac{1}{2}\right) \\ &-\frac{1}{2}\operatorname{sign}\left(y + 2m - \frac{1}{2}\right). \end{aligned}$$

The function $M^m(y)$ has the following useful property, which is immediate from the definition of M^m.

LEMMA 3.1. *If $m \le |y| < 2m$, then $M^m(y) = \operatorname{sign}(y)$, and otherwise $M^m(y) = 0$.*

Let us look at a fixed input x. Assume that $2^l \le |F(x)| < 2^{l+1}$. Given this information, we can use less precision in the weights w_i. Set

$$F_{(l)}(x) = \lfloor w_0 m/2^l \rfloor + \sum_{i=1}^{n} \lfloor w_i m/2^l \rfloor x_i.$$

If we disregard the error introduced by the floor operation, we will have

$$m \le |F_{(l)}(x)| < 2m.$$

It is plausible that for most inputs x the truncation error will not matter, and then $\operatorname{sign}(F(x)) = \operatorname{sign}\big(F_{(l)}(x)\big)$. To get the weights small, we just look at $F_{(l)}(x)$ modulo some small prime $p > m^2$.

We are now ready to construct a small gadget $\varphi^{(l)}(x)$ that implements the computation described above.

$$\begin{aligned} w_i^{(l)} &= \lfloor w_i m/2^l \rfloor \operatorname{rem} p, \\ \Phi_{(l)}(x) &= w_0^{(l)} + \sum_{i=1}^{n} w_i^{(l)} x_i, \\ \varphi_{(l)}(x) &= \sum_{j=-\lfloor (n+1)/2 \rfloor}^{\lceil (n+1)/2 \rceil} M^m\big(\Phi_{(l)}(x) + jp\big). \end{aligned} \tag{3}$$

Let us establish some properties of $\varphi_{(l)}$ as defined by (3).

PROPOSITION 3.2. *For $p > 4m$ the following holds.*

1. *For any x, $\big|\varphi_{(l)}(x)\big| \in \{0, 1\}$.*
2. *If $\big|F_{(l)}(x) \operatorname{rem} p\big| \notin [m, 2m)$, then $\varphi_{(l)}(x) = 0$.*
3. *If $m \le \big|F_{(l)}(x)\big| < 2m$, then $\varphi_{(l)}(x) = \operatorname{sign}\big(F_{(l)}(x)\big)$.*

Proof. All statements follow from Lemma 3.1.

1. Each term $M^m\left(\Phi_{(l)}(x)+jp\right)$ is either 0, 1, or -1, and since $p>4m$ there is at most one j for which $\left|\Phi_{(l)}(x)+jp\right|<2m$, and thus at most one nonzero term $M^m\left(\Phi_{(l)}(x)+jp\right)$.
2. By Lemma 3.1, $M^m(y)\neq 0$ implies that $|y|<2m$, and since $p>4m$ this in turn implies $y=y \text{ rem } p$. Thus, the only term that might be nonzero is the term $M^m\left(\Phi_{(l)}(x)+jp\right)$ for which $\Phi_{(l)}(x)+jp=\Phi_{(l)}(x) \text{ rem } p$. Now just observe that $F_{(l)}(x)\equiv\Phi_{(l)}(x)(\text{mod } p)$.
3. For the third statement we have the following. Assume that $m\leq\left|F_{(l)}(x)\right|<2m$. We then have $\Phi_{(l)}(x) \text{ rem } p=F_{(l)}(x)$.
   Let j_0 be the integer that satisfies $\Phi_{(l)}(x) \text{ rem } p=\Phi_{(l)}(x)+j_0p$.
   Since $\left|\Phi_{(l)}(x)\right|\leq(n+1)(p-1)/2$, j_0 occurs in the sum in (3). The third statement now follows from Lemma 3.1. □

It is tempting to set

$$\varphi(x)=\sum_l \varphi_{(l)}(x) \tag{4}$$

and hope that $\varphi(x)=\text{sign}\,(F(x))=f(x)$. The idea is that there is always an l such that $2^l\leq|F(x)|<2^{l+1}$. If there was no error introduced by the floor operations, we would have $m\leq\left|F_{(l)}(x)\right|<2m$ and $\text{sign}\left(F_{(l)}(x)\right)=\text{sign}\,(F(x))$. By Proposition 3.2, $\varphi_{(l)}(x)=\text{sign}\left(F_{(l)}(x)\right)$, and for all other l we would probably have $\varphi_{(l)}(x)=0$. Then we would have $\varphi(x)=\varphi_{(l)}(x)=\text{sign}\,(F(x))=f(x)$.

Remark 2. We know that $|F(x)|\leq(n+1)W$. Thus, it is sufficient to sum over l such that $0\leq l\leq\log((n+1)W)$ in (4).

The problems with (4) are of course that the floor operation sometimes introduces *truncation errors* (e.g., when $2^l\leq F(x)$ but $F_{(l)}(x)<m$), and sometimes the "wrong" l gives a nonzero contribution to the sum (a *modular error*). Equation (4) is not such a bad idea though, because it works for most x.

4. The approximator. To get an approximator based on the ideas of the previous section we want to spread the value $F(x)$ in some random fashion. This was done in [10] by considering $\text{sign}\,(\alpha F(x))$ for a randomly chosen integer $\alpha\in\{1,2,\ldots,2^{2n}\}$, and since $\alpha>0$, one has $\text{sign}\,(F(x))=\text{sign}\,(\alpha F(x))$. It was shown that for any x, the approximator would compute $\text{sign}\,(\alpha F(x))$ correctly with high probability. By taking many independent α's and taking the sign of the average, one gets an approximator that behaves well on all inputs. If the range of α was smaller, we could get an explicit approximator by taking the average over all α's.

We will modify the construction. Our approximator will depend on two parameters, and for any input, the approximator will be good with high probability if the parameters are chosen at random. In our case the probability space is small enough that we may take the average over all possible choices of parameter values. Just like in [10] we will use a multiplier α, but it will be much smaller. For a random α, the probability of a truncation error is small. To handle the modular errors, we will also choose the prime p, used in the construction, randomly, and the probability that we get a modular error for a random p is small.

DEFINITION 4.1. *We define the following sets. Let*

$$[m]=\{1,2,\ldots,m\},$$

$\mathcal{PR}^m$ *is the set of the first* m^2 *primes that are greater than* m^2.

When α *and* p *are chosen randomly, they are picked as a pair* $(\alpha,p)\in[m]\times\mathcal{PR}^m$ *according to the uniform distribution.*

Instead of looking at the weights w_i, we make the approximator for weights αw_i. Thus, instead of looking at $F(x)$ we look at $\alpha F(x)$, but this works since $\operatorname{sign}(\alpha F(x)) = \operatorname{sign}(F(x)) = f(x)$. We call the corresponding truncated linear form $F^{\alpha}_{(l)}(x)$; that is,

$$F^{\alpha}_{(l)}(x) = \lfloor \alpha w_0 m/2^l \rfloor + \sum_{i=1}^{n} \lfloor \alpha w_i m/2^l \rfloor x_i.$$

Now we are ready to define the parametrized approximator $\varphi^{\alpha,p}(x)$.

$$\begin{aligned} w_i^{(l)} &= \left\lfloor \alpha w_i m/2^l \right\rfloor \operatorname{rem} p, \\ \Phi^{\alpha,p}_{(l)}(x) &= w_0^{(l)} + \sum_{i=1}^{n} w_i^{(l)} x_i, \\ \varphi^{\alpha,p}_{(l)}(x) &= \sum_{j=-\lfloor (n+1)/2 \rfloor}^{\lceil (n+1)/2 \rceil} M^m(\Phi^{\alpha,p}_{(l)}(x) + jp), \qquad (5) \\ \varphi^{\alpha,p}(x) &= \sum_{l=0}^{\lfloor 3 \log W \rfloor + 1} \varphi^{\alpha,p}_{(l)}(x), \end{aligned}$$

where the upper bound of the last summation follows from an argument analogous to Remark 3, assumptions (1) and (2), and the fact that $\alpha \le m$.

The following proposition is the (α, p)-version of Proposition 3.2. The proof is completely analogous.

PROPOSITION 4.2. *For $p > m^2$ the following hold.*

1. *For any x, $\left|\varphi^{\alpha,p}_{(l)}(x)\right| \in \{0, 1\}$.*
2. *If $\left|F^{\alpha}_{(l)}(x) \operatorname{rem} p\right| \notin [m, 2m)$ then $\varphi^{\alpha,p}_{(l)}(x) = 0$.*
3. *If $m \le \left|F^{\alpha}_{(l)}(x)\right| < 2m$ then $\varphi^{\alpha,p}_{(l)}(x) = \operatorname{sign}\big(F^{\alpha}_{(l)}(x)\big)$.*

COROLLARY 4.3. *If we assume* (1) *and* (2)*, then for all $x \in \mathbb{Z}^n$ we have $\varphi^{\alpha,p}(x) \in \mathbb{Z}$ and $|\varphi^{\alpha,p}(x)| \le 2 + 3 \log W$.*

Proof. It is not hard to see that statement 4.2 of Proposition 4.2 holds even for $x \in \mathbb{Z}^n$. The corollary follows since we sum over at most $2 + 3 \log W$ different l in (5). □

For any x there are usually α's and p's such that we get truncation errors or modular errors. We will show that for any fixed x most pairs (α, p) do not give such errors.

DEFINITION 4.4.

1. *We say that α is* bad *for x if there is some l such that $\big||\alpha F(x)| - 2^l\big| \le 2^{l+1} n/m$. Otherwise α is* good *for x.*
2. *We say that p is* bad *for x and α if there is an l such that $\left|F^{\alpha}_{(l)}(x)\right| \ge 2m$ but $\left|F^{\alpha}_{(l)}(x) \operatorname{rem} p\right| < 2m$. Otherwise p is* good *for x and α.*
3. *We call a pair (α, p)* bad *for x if α is bad for x or if p is bad for x and α. Otherwise (α, p) is* good *for x.*

We will show that when a pair (α, p) is good for x, then $\varphi^{\alpha,p}(x) = f(x)$. We will also show that for any fixed x, a random pair (α, p) is likely to be good.

First we show that $\varphi^{\alpha,p}(x) = f(x)$ when (α, p) is good.

PROPOSITION 4.5. *Let x be an arbitrary fixed input. If (α, p) is good for x, then $\varphi^{\alpha,p}(x) = f(x)$.*

Proof. Let x be a fixed input and assume that (α, p) is a good pair for x. Without loss of generality, we assume that $F(x) > 0$.

First we will show that there are no truncation errors that matter in this case. Let l_0 be the integer such that

$$2^{l_0} \leq \alpha F(x) < 2^{l_0+1}.$$

Since $F(x) \leq (n+1)W$ and $\alpha \leq m$ we have $\alpha F(x) \leq m(n+1)W \leq 2W^3$, and thus $l_0 \leq \lfloor 3 \log W \rfloor + 1$.

For any l

$$\left| F^{\alpha}_{(l)}(x) - \frac{\alpha F(x) m}{2^l} \right| < n+1, \tag{6}$$

since the floor operation makes an error less than 1 for each weight of the gate.

Equation (6) immediately tells us that

$$m < F^{\alpha}_{(l_0)}(x) < 2m.$$

This implies that the term $\varphi^{\alpha,p}_{(l_0)}(x)$ will be equal to 1.

For $l \neq l_0$ we have the following. Let $l < l_0$ be an arbitrary integer. Since α is good for x we have

$$2^{l+1}\left(1 + \frac{2n}{m}\right) < \alpha F(x),$$

and from (6) it follows that

$$2m < F^{\alpha}_{(l)}(x).$$

A completely analogous argument shows that for any $l > l_0$ one has

$$m > F^{\alpha}_{(l)}(x).$$

Thus, no error is introduced due to truncation.

We still need to show that there are no modular errors, that is, when we do the computation modulo p we do not get a contribution from some $l > l_0$ for which $F^{\alpha}_{(l)}(x) > 2m$. However, this would mean that for some l one has $F^{\alpha}_{(l)}(x) > 2m$ but $\left|F^{\alpha}_{(l)}(x) \text{ rem } p\right| < 2m$, and this implies by definition that p is bad for x and α contrary to our assumption. □

It remains to show that for any fixed x most α's are good, and for any fixed x and α most p's are good.

LEMMA 4.6. *If W and m satisfy* (1) *and* (2), *then for any x*

$$\Pr\alpha \textit{ is bad for } x < (16 \log^2 \mathrm{W})/m.$$

Proof. Let $r = |F(x)|$. We want to show that for most α we have $\left|2^l - \alpha r\right| > 2^{l+1} n/m$ for all l.

Look at a fixed l. If there is any α which is bad with respect to l, then r must satisfy the following equations.

$$r \leq 2^l\left(1 + \frac{2n}{m}\right) < 2^{l+1}, \tag{7}$$

$$r \geq \frac{2^l}{m}\left(1 - \frac{2n}{m}\right) > 2^{l-1}/m. \tag{8}$$

The bad interval for α has length $2^{l+2}n/(rm)$. If r and l do not satisfy (8), then all α's are too small to be bad. On the other hand, if r and l satisfy (8), there are at most $8n$ different α's that fit in the bad interval.

For any r, there are less than $2\log m$ different l that satisfy (7) and (8). Thus, any x has at most $16n\log m$ bad α. The lemma follows by our assumption that (1) and (2) hold. □

LEMMA 4.7. *If W and m satisfy* (1) *and* (2)*, then for any x and α*

$$\Pr p \text{ is bad for } x \text{ and } \alpha < (16\log^2 W)/m.$$

Proof. Once again consider a fixed l first, and assume that for input x and multiplier α we have $|F^{\alpha}_{(l)}(x)| \geq 2m$ but $|F^{\alpha}_{(l)}(x) \operatorname{rem} p| < 2m$ for $4m\log W$ of the primes. By the pigeon hole principle we must have more than $\log W$ primes that give the same remainder k, such that $|k| < 2m$. By the Chinese remainder theorem, $F^{\alpha}_{(l)}(x)$ is uniquely determined modulo the product of those primes. That product is greater than $m^{2\log W}$. Since we assume (1) and (2),

$$\begin{aligned}\left|F^{\alpha}_{(l)}(x)\right| &\leq |m\alpha F(x)| \\ &\leq m^2(n+1)W \\ &\ll m^{2\log W}.\end{aligned}$$

This implies that $k = F^{\alpha}_{(l)}(x)$, which means that $\left|F^{\alpha}_{(l)}(x)\right| < 2m$, and we have a contradiction. Since there are at most $2 + 3\log W < 4\log W$ different l, the lemma follows. □

Lemmas 4.6 and 4.7 imply the following lemma.

LEMMA 4.8. *If W and m satisfy* (1) *and* (2)*, then for an arbitrary fixed input x,*

$$\Pr(\alpha, p) \text{ is bad for } x \leq (32\log^2 W)/m.$$

We conclude this section by showing how the approximator allows us to simulate a single threshold gate (with large weights) by a depth-2, small-weight, threshold-circuit. This is a constructive version of Theorem 7.8 from [10].

THEOREM 4.9. $LT[1] \subsetneq \widehat{LT}[2]$.

Proof. We will prove the inclusion. That it is strict follows from the fact that parity is in $\widehat{LT}[2] \setminus LT[1]$. Assume $n \geq 100$; smaller inputs are handled by writing the function in disjunctive normal form.

We are given a threshold gate

$$f(x) = \operatorname{sign}\left(w_0 + \sum_{i=1}^{n} w_i x_i\right).$$

Let $W = \max\{w_{\max}(f), 2^n\}$ be the bound used in the construction, and set the parameter $m = 128\log^3 W$ (observe that W and m satisfy (1) and (2)). Consider the following function:

$$g(x) = \operatorname{sign}\left(\sum_{(a,p)} \varphi^{\alpha,p}(x)\right).$$

We claim two things: $f(x) = g(x)$ for all $x \in \{1,-1\}^n$, and $g(x)$ can be computed by a depth-2, polynomial-size, threshold circuit with polynomially bounded weights.

The correctness of the construction follows from

$$E[f(x)\varphi^{\alpha,p}(x)] > 0, \tag{9}$$

where x is arbitrary and fixed, and we take expectation over the uniform distribution on pairs (α, p). Let us prove (9).

$$\begin{aligned} &E[f(x)\varphi^{\alpha,p}(x)] \\ &\quad \geq \Pr(\alpha,p) \text{ good} \\ &\qquad -(2+3\log W)\Pr(\alpha,p) \text{ bad} \\ &\quad \geq 1 - \frac{1}{4\log W} - \frac{2+3\log W}{4\log W} \\ &\quad > 0, \end{aligned}$$

where the first inequality follows from Corollary 4.3 and the second inequality follows from Lemma 4.8. This shows that $f(x) = g(x)$ for all x.

Now, to implement $g(x)$ as a circuit, let us look at what $g(x)$ does.

$$g(x) = \operatorname{sign}\left(\sum_{\alpha,p,l,j} \frac{1}{2}\operatorname{sign}\left(\Phi_{(l)}^{\alpha,p}(x) + jp - m + \frac{1}{2}\right) - \frac{1}{2}\operatorname{sign}\left(\Phi_{(l)}^{\alpha,p}(x) + jp - 2m + \frac{1}{2}\right)\right.$$
$$\left. + \frac{1}{2}\operatorname{sign}\left(\Phi_{(l)}^{\alpha,p}(x) + jp + m - \frac{1}{2}\right) - \frac{1}{2}\operatorname{sign}\left(\Phi_{(l)}^{\alpha,p}(x) + jp + 2m - \frac{1}{2}\right)\right).$$

$\Phi_{(l)}^{\alpha,p}(x)$ is simply a weighted sum of the inputs. Hence, we only have two levels of sign-functions, and one is the outmost operator, so $g(x)$ can be computed by a depth-2 threshold circuit. The size is the total number of terms in the summations (recall that each $\Phi_{(l)}^{\alpha,p}(x)$ depends on $n \leq \log W$ variables). There are $m^3 = O\left(\log^9 W\right)$ pairs (α, p), and $O(\log W)$ different l. This gives a total size of $O(\log^{12} W)$.

As for the weights, they are not necessarily integers, but if we multiply them by 2 they are. The only weights that are not ± 1 are of the form $2w_j^{(l)}$, $2(w_0^{(l)} + jp \pm m)$, or $2(w_0^{(l)} + jp \pm 2m)$. All the $w_i^{(l)}$ are reduced modulo some prime p that is among the first $2m^2 = O(\log^6 W)$ primes. This implies that $p \leq O(\log^7 W)$. Since $-\lfloor (n+1)/2 \rfloor \leq j \leq \lceil (n+1)/2 \rceil$, the weights are all of magnitude $O(\log^8 W)$.

If we assume Muroga's bound on weights to hold for the original threshold gate f, then $\log W \leq O(n \log n)$, and hence we have a polynomial-size, polynomial-weight threshold circuit of depth 2 that computes $f(x)$. Actually, we need only to have $\log W$ polynomial in n. □

5. Extending the construction to circuits. In the previous section we showed that a single threshold gate can be simulated by a polynomial-size majority circuit. We will now generalize this to show that a depth-d, polynomial-size threshold circuit can be simulated by a depth-$d+1$, polynomial-size majority circuit.

In [10] Goldmann, Håstad, and Razborov introduced a circuit class that mixes small and large weights.

DEFINITION 5.1. *$\widetilde{LT}_d$ is the class of functions computable by depth-d, polynomial-size circuits where the top gate has polynomially bounded weights.*

We will show the following theorem.

THEOREM 5.2. *For any depth d, possibly depending on n, $\widetilde{LT}_d = \widehat{LT}_d$. Moreover, given an $\widetilde{LT}_d$ circuit, for input size n, with weights bounded by $2^{p(n)}$ for some polynomial p, there is an explicit $\widehat{LT}_d$ circuit that computes the same function.*

COROLLARY 5.3. *For any depth d, possibly depending on n, $LT_d = \widehat{LT}_d + 1$. Moreover, given an LT_d circuit, for input size n, with weights bounded by $2^{p(n)}$ for some polynomial p, there is an explicit $\widehat{LT}_d + 1$ circuit that computes the same function.*

Proof. Make the LT_d circuit into an $\widetilde{LT}_d + 1$ circuit by adding a dummy gate with the output-gate of the LT_d circuit as its only input and give this input weight 1. By Theorem 5.2 the $\widetilde{LT}_d + 1$ circuit can be turned into an $\widehat{LT}_d + 1$ circuit. □

It remains to prove the theorem.

Proof of Theorem 5.2. Let us look at a circuit C for an $\widetilde{LT}_d$-function. For simplicity, assume that the weights at the top gate are all 1. Let the top gate be given by

$$g(x) = \operatorname{sign}\left(\sum_{i=1}^{t} C_i(x)\right),$$

where the C_i are depth-$d-1$ threshold circuits. Let s be the size of C. For each circuit C_i we construct an approximator $\Gamma_i^{\alpha,p}$. Below we describe how this is done.

Let $w_{\max}(C)$ be the largest magnitude of a weight of the circuit C, and let $W = \max\{w_{\max}(C), 2^s\}$. Let C_k be one of the subcircuits of g. Let the gates of C_k be $f_1, \ldots, f_r$, where $r \le s$. The fan-in of any f_i is trivially bounded by $\log W$, and all gates have their weights bounded by W. Set the parameter m of the previous section as follows:

$$m = 256 s\, t \log^3 W.$$

For $(\alpha, p) \in [m] \times \mathcal{PR}^m$, construct a gate approximator $\varphi_i^{\alpha,p}$ for each gate f_i. Now, connect the approximators in the same way as the gates. That is, if the output of f_i is the kth input to f_j, then the output of $\varphi_i^{\alpha,p}$ is the kth input to $\varphi_j^{\alpha,p}$. We call the constructed "circuit" $\Gamma_k^{\alpha,p}$. Note that for any fixed input x, if (α, p) is simultaneously good for x in all $\varphi_i^{\alpha,p}$ of $\Gamma_k^{\alpha,p}$, then $\Gamma_k^{\alpha,p}(x) = C_k(x)$.

LEMMA 5.4. *For any fixed input x, if (α, p) is chosen uniformly at random from $[m] \times \mathcal{PR}^m$, then* $\mathrm{Pr}\Gamma_{\mathrm{k}}^{\alpha,\mathrm{p}}(\mathrm{x}) \neq \mathrm{C_k(x)} \le (8\mathrm{t}\log \mathrm{W})^{-1}$.

Proof. We know that each gate f_i of C_k has fan-in bounded by $\log W$, and will give the correct output if (α, p) is good. By Lemma 4.8, the probability that (α, p) is bad is $(8s\,t \log W)^{-1}$ by the choice of m. Since there are at most s gates in C_k, the lemma follows. □

We say that (α, p) is good for x and $\Gamma_k^{\alpha,p}$ if it is good for all gate approximators $\varphi_i^{\alpha,p}$ of $\Gamma_k^{\alpha,p}$ simultaneously.

What happens when (α, p) is bad for $\Gamma_k^{\alpha,p}$? The magnitude of the output of $\Gamma_k^{\alpha,p}$ is bounded by the maximum output of the approximator of the top gate of C_k. Using Corollary 4.3, we have for all x and all (α, p),

$$|\Gamma_k^{\alpha,p}(x)| \le 2 + 3 \log W. \tag{10}$$

Since the corollary holds even when the inputs to the top gate are integers, this bound holds even when some other approximator in $\Gamma_k^{\alpha,p}$ fails.

If we combine Lemma 5.4 and (10) we get the following lemma.

LEMMA 5.5. *For any fixed input x, if (α, p) is chosen uniformly at random from $[m] \times \mathcal{PR}^m$, then $|\mathrm{E}[\Gamma_{\mathrm{k}}^{\alpha,\mathrm{p}}(\mathrm{x})] - \mathrm{C_k(x)}| \leq \frac{1}{2t}$.*

Do the same for all the circuits C_i to get approximators $\Gamma_i^{\alpha,p}$. Consider the following function:

$$h(x) = \operatorname{sign}\left(\sum_{i=1}^{t} \sum_{(\alpha,p)} \Gamma_i^{\alpha,p}(x)\right).$$

Observe that this is equivalent to

$$h(x) = \operatorname{sign}\left(E\left[\sum_{i=1}^{t} \Gamma_i^{\alpha,p}(x)\right]\right). \tag{11}$$

We claim the following.

PROPOSITION 5.6. *For all inputs x, $h(x) = g(x)$.*

Proof. By (11) and the fact that $\left|\sum_{i=1}^{t} C_i(x)\right| \geq 1$, it is sufficient to show that

$$\left|\mathrm{E}\left[\sum_{i=1}^{t} \Gamma_i^{\alpha,p}(x)\right] - \sum_{i=1}^{t} C_i(x)\right| \leq 1/2.$$

This inequality follows by straightforward application of Lemma 5.5. □

An argument analogous to that in the previous section shows that $g(x)$ can be implemented as a depth-d threshold circuit. We call the circuit $\widehat{C}$. It is not hard to see that each wire in C corresponds to polynomially (in $\log W$) many wires in $\widehat{C}$.

To get integer weights in the circuit it is sufficient to multiply all weights by 2. The weights all have magnitude bounded by order of the largest prime used multiplied by s. The largest prime in $\mathcal{PR}^m$ has size $O(m^2 \log m)$, so the weights are all $O(s^3 t^2 \log^6 W(\log(st) + \log\log W))$. This completes the proof of Theorem 5.2.

Just as before, the construction is polynomial in n as long as $\log W$ is polynomial in n. By Theorem 2.1, this is always possible. □

6. Uniform computations. In addition to being explicit, the constructions given in this paper also preserve the uniformity conditions of the classes of circuits. The constructions of sections 4 and 5 yield the following.

THEOREM 6.1. *Given any q-uniform class of threshold circuits $\{C_n\}$ of depth d and polynomial size, $q \in \{L, P\}$. There exists an algorithm running in logspace that, given a description of a circuit C_n of depth d and polynomial size, will output the description of a small-weight threshold circuit C'_n of depth $d+1$ and a polynomial size such that $f_{C_n} = f_{C'_n}$ for all $n \geq 0$.*

We have also the following corollary.

COROLLARY 6.2.

$$L\text{-uniform } LT_d \subseteq L\text{-uniform } \widehat{LT}_{d+1},$$
$$P\text{-uniform } LT_d \subseteq P\text{-uniform } \widehat{LT}_{d+1}.$$

Proof. For P-uniformity the proof is very simple: generate the large-weight circuit and then translate it to a small-weight circuit using Theorem 6.1.

To prove the result for L-uniformity we use the following lemma.

LEMMA 6.3. *A family of circuits is L-uniform if and only if there is a Turing machine that on input n outputs $\overline{C}_n$ using space $O(\log n)$.*

Proof. Assume that we have an L-uniform family of circuits $\mathcal{C}$, that is, we have a machine M that recognizes the language $L_{\mathcal{C}}$. Let $\overline{C}_n$ be the description of the circuit for input size n.

There are constants c and c' such that each $(n, g) \in L_G$ has at most $c(1+\log(n+1))$ bits, and each $(n, e, v, g, k, \delta) \in L_E$ has at most $c'(1 + \log(n + 1))$ bits. Consider the machine M' that, given n, runs through all possible tuples with n as the first component and outputs those that belong to $L_{\mathcal{C}}$. This will output $\overline{C}_n$. The space required is $O(\log n)$ plus the space used by M and the time is polynomial in n times the time used by M. Thus, we have proved the "only if" part of the lemma.

Next we consider the case when there is a machine M that on input n generates $\overline{C}_n$. Let us construct a machine M' that works as follows. When given an input tuple, say (n, g) for instance, it checks if $|(n, g)| > c(1 + \log(n + 1))$ and, if so, it rejects; otherwise it runs M on input n and checks the output of M "on-line," and if (n, g) is produced then M' accepts; otherwise it rejects. Clearly, M' recognizes $L_{\mathcal{C}}$. Also, M uses space $O(\log n)$ plus the space used by M and time polynomial in n plus the time used by M. We have now proved the "if" part of both statements and the proof of the lemma is complete. □

The following argument completes the proof of the corollary. If a family of (large-weight) threshold circuits is L-uniform, then by Lemma 6.3 there is a machine that generates $\overline{C}_n$ in space $O(\log n)$. By Theorem 6.1 this description can be translated to a description of an equivalent small-weight circuit in space $O(\log n)$. Applying the lemma again, in the other direction, we see that the family of small-weight circuits is also L-uniform. □

It remains to prove Theorem 6.1. Below we explain how, given a polynomial-size description of a large-weight threshold circuit, one can construct a description of an equivalent small-weight threshold circuit with one extra level of gates. It will be clear that all the necessary computations can be carried out in space $O(\log n)$. In order to conform with the notation used so far, we allow weights of the form $w/2$, where w is an integer. However, as stated before, if all weights in the circuit are multiplied by 2, then the circuit has integral weights.

In what follows

- α ranges over $[m]$,
- p ranges over $\mathcal{PR}^m$,
- l ranges from 0 to $\lfloor 3 \log W \rfloor + 1$,
- j ranges from $-\lfloor (n+1)/2 \rfloor$ to $\lceil (n+1)/2 \rceil$,
- β ranges over $\{-1, 1\}$,
- γ ranges over $\{1, 2\}$,
- g is a gate label in a large-weight circuit,
- and e is an edge label in a large-weight circuit.

The parameters α through γ correspond to the threshold gates in an approximator for a large-weight gate, since the approximator can be written on the following form (for fixed α and p):

$$\sum_{l,j,\beta,\gamma} \frac{3 - 2\gamma}{2} \operatorname{sign}\left(\Phi_{(l)}^{\alpha,p} + jp + \beta\gamma m - \frac{\gamma}{2}\right). \tag{12}$$

Note that the summation over β and γ corresponds to the four terms in the function M^m. The idea now is that for each (large-weight) gate in the original circuit, we

construct the collection of gates in (12), and for each edge between two gates in the original circuit we construct edges between the corresponding sets of small-weight gates in the new (small-weight) circuit.

We use g and e for gate labels and edge labels of the large-weight circuit, and we use $\hat{g}$ and $\hat{e}$ to denote labels in the small-weight circuit. To describe the construction it is convenient to define *labeling functions*. We have functions $\lambda_g(n, g, \alpha, p, l, j, \beta, \delta)$ (to generate new gate labels), $\lambda_e(n, e, \hat{g}', \hat{g})$ (to generate new edge labels), $\lambda_1(n, \hat{g})$, and $\lambda_t(n, \hat{g})$ (which also generate edge labels). We also use $\hat{g}_t$ as the label of the top gate of the new circuit. We will assume that

1. the functions are linear space computable (i.e., space $O(\log n)$),
2. the size of the output is linear in the size of the input (i.e., size $O(\log n)$),
3. the functions λ_e, λ_1, and λ_t have pairwise disjoint images,
4. the set I_n is disjoint from the image of λ_g,
5. $\hat{g}_t$ is not in I_n or in the image of λ_g,
6. the functions are one-to-one.

Such functions are easy to find and we will not concern ourselves with the exact implementation of them.

The translation works as follows:

1. Compute the size s of the large-weight circuit and find the largest k-value of any edge and call this $k_{\max}$.
2. Compute $L = \max\{s, k_{\max}\}$ (note that $W = 2^L$).
3. Compute $m = 256sL^3$.
4. Output $(n, \hat{g}_t)$.
5. For each gate (n, g) we output a collection of gates that corresponds to the threshold functions of (12) for each value of α and p:

```
for each (α, p) ∈ [m] × PR^m
    for each l ∈ {0, 1, ..., 3L + 1}
        for each j ∈ {−⌊(n + 1)/2⌋, ..., ⌈(n + 1)/2⌉}
            for each (β, γ) ∈ {−1, 1} × {1, 2}
                ĝ ← λ_g(n, g, α, p, l, j, β, γ)
                output (n, ĝ)
                ê ← λ_1(n, ĝ)
                w ← jp + βγm − β/2
                output (n, ê, 1, ĝ, |w|, sign(w))
```

6. For each edge (n, e, v, g, k, δ) construct a collection of new edges. How this is done depends on whether v is an input or a gate.
   If v is an input, then it must be fed, with the appropriate weight, to all of the gates that are constructed from g in the previous step.

```
for each (α, p) ∈ [m] × PR^m
    for each l ∈ {0, 1, ..., 3L + 1}
        for each j ∈ {−⌊(n + 1)/2⌋, ..., ⌈(n + 1)/2⌉}
            for each (β, γ) ∈ {−1, 1} × {1, 2}
                ĝ ← λ_g(n, g, α, p, l, j, β, γ)
                ê ← λ_e(n, e, v, ĝ)
                w ← ⌊αγδm2^{k−l}⌋ rem p
                output (n, ê, v, ĝ, |w|, sign(w))
```

   If, on the other hand, we have an edge (n, e, g', g, k, δ) between two gates, the situation is quite similar, except that many of the gates that g' gives rise to are connected to many of the gates that g gives rise to.

```
for each (α, p) ∈ [m] × PR^m
```

```
for each (l, l') ∈ {0, 1, ..., 3L + 1}^2
    for each (j, j') ∈ {−⌊(n + 1)/2⌋, ..., ⌈(n + 1)/2⌉}^2
        for each (β, β', γ, γ') ∈ {−1, 1}^2 × {1, 2}^2
            ĝ ← λ_g(n, g, α, p, l, j, β, γ)
            ĝ' ← λ_g(n, g', α, p, l', j', β', γ')
            ê ← λ_e(n, e, ĝ', ĝ)
            w ← ⌊α(3 − 2γ)γδm2^{k−l−1}⌋ rem p
            output (n, ê, ĝ', ĝ, |w|, sign(w))
```

7. Finally, let g_t be the top gate of the original circuit. Each gate that is produced from g_t in step 5 is to be fed with weight 1 to $\hat{g}_t$, the top gate of the new circuit.

```
for each (α, p) ∈ [m] × PR^m
    for each l ∈ {0, 1, ..., 3L + 1}
        for each j ∈ {−⌊(n + 1)/2⌋, ..., ⌈(n + 1)/2⌉}
            for each (β, γ) ∈ {−1, 1} × {1, 2}
                ĝ ← λ_g(n, g_t, α, p, l, j, β, γ)
                ê ← λ_t(n, ĝ)
                output (n, ê, ĝ, ĝ_t, 1, 1)
```

Except for the calculation of w in step 6 it is immediately clear that the computations involved can be carried out in space $O(\log n)$. This is true also for the weight computation. For $\lfloor \alpha\gamma\delta m 2^{k-l} \rfloor$ rem p, note that $\alpha\gamma m$ can be represented with $O(\log n)$ bits, and that when $k < l$ then so can $\lfloor \alpha\gamma\delta m 2^{k-l} \rfloor$. When $k \geq l$, then $\alpha\gamma\delta m 2^{k-l}$ is an integer, and computing 2^{k-l} rem p is easily done in space $O(p + k + l) = O(\log n)$.

7. Conclusions and open problems. Our results entail the first explicit constructions for the optimal-depth, polynomial-size majority circuits for the number of basic functions including, among others, *powering* (depth 3), *integer multiplication* and *integer division* (depth 3); see [27] and [4].

More generally, our results entail the *uniformity* of the classes of majority circuits simulating the corresponding classes of *threshold circuits*. We look at the following functions.

ADDITION: given two n-bit numbers, compute their sum.

MULTIPLE ADDITION: given n n-bit numbers, compute their sum.

MULTIPLICATION: given two n-bit numbers, compute their product.

MULTIPLE MULTIPLICATION: given n n-bit numbers, compute their product.

DIVISION: given a $2n$-bit numbers x and an n-bit number y, compute $\lfloor x/y \rfloor$.

SQUARING: given an n-bit number x, compute x^2.

POWERING: given an n-bit number x and a $\log n$-bit number y, compute x^y.

COMPARISON: given two n-bit numbers x and y, decide if $x \geq y$.

MAXIMUM: given n n-bit numbers, output the largest one.

SORTING: given n n-bit numbers, output them in sorted order.

The following table surveys the uniform upper bounds known before and stemming from the present paper and compares them with the best-known (nonuniform) lower bounds. Most of the constructions follow from nonuniform versions in [27] and [26] (see also [22]). The uniform constructions for ADDITION, COMPARISON, and SORTING follow from [3].

Function	Uniform Depth Upper Bound	Lower Bound
ADDITION	2,L-uniform	2
MULTIPLE ADDITION	2,L-uniform	2
MULTIPLICATION	3,L-uniform	3 [11]
MULTIPLE MULTIPLICATION	4,L-uniform	3 [11]
DIVISION	3,P-uniform	3 [33]
SQUARING	3,L-uniform	3 [33]
POWERING	3,P-uniform	2 [33]
COMPARISON	2,L-uniform	2 [25]
MAXIMUM	3,L-uniform	2
SORTING	3,L-uniform	3 [26]

We conclude with a list of open problems:

1. If the original circuit is monotone, our construction yields a nonmonotone majority circuit. It is an open question if an arbitrary monotone threshold gate can be simulated by constant-depth monotone majority circuits of polynomial size.
2. Alternating Turing machines are closely connected to circuits with AND-gates and OR-gates, and counting Turing machines are connected to majority circuits. Is there a reasonable machine model that has such a relationship to threshold circuits? If so, what would be the corresponding notion of uniformity, and would our simulation still work?
3. Are there any strong lower bounds for depth-2 threshold circuits or depth-3 majority circuits computing some explicit function?

Acknowledgments. We are grateful to Johan Håstad and Ingo Wegener for many valuable comments on earlier versions of this paper. We also thank Jens Lagergren, Sasha Razborov, and Avi Wigderson for several helpful discussions on the topic of this paper. Finally, we thank the referees for careful reading and for numerous helpful suggestions and comments.

REFERENCES

[1] E. Allender, *A note on the power of threshold circuits*, in Proc. 30th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1989, pp. 580–584.

[2] N. Alon and R. B. Boppana, *The monotone circuit complexity of boolean functions*, Combinatorica, 7 (1987), pp. 1–22.

[3] N. Alon and J. Bruck, *Explicit Constructions of Depth-2 Majority Circuits for Comparison and Addition*, Tech. Report RJ 8300 (75661), IBM Research Division, 1991. SIAM J. Discrete Math., 7 (1994), pp. 1–8.

[4] P. W. Beame, S. A. Cook, and H. J. Hoover, *Log depth circuits for division and related problems*, SIAM J. Comput., 15 (1986), pp. 994–1003.

[5] R. Beigel and J. Tarui, *On ACC*, in Proc. 32nd IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1991, pp. 783–792.

[6] J. Bruck, *Harmonic analysis of polynomial threshold functions*, SIAM J. Discrete Math., 3 (1990), pp. 168–177.

[7] J. Bruck and R. Smolensky, *Polynomial threshold functions, AC^0 functions and spectral norms*, in Proc. 31st IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 632–641.

[8] A. Chandra, L. Stockmeyer, and U. Vishkin, *Constant depth reducibility*, SIAM J. Comput., 13 (1984), pp. 423–439.

[9] S. A. Cook, *A taxonomy of problems with fast parallel algorithms*, Inform. and Control, 64 (1985), pp. 2–22.

[10] M. Goldmann, J. Håstad, and A. Razborov, *Majority gates versus general weighted threshold gates*, Comput. Complexity, 2 (1992), pp. 277–300.
[11] A. Hajnal, W. Maass, P. Pudlák, M. Szegedy, and G. Turán, *Threshold circuits of bounded depth*, in Proc. 28th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1990, 1987, pp. 99–110.
[12] J. Håstad, *On the size of weights for threshold gates*, SIAM. J. Disc. Math., 7 (1994), pp. 484–492.
[13] J. Håstad and M. Goldmann, *On the power of small-depth threshold circuits*, Comput. Complexity, 1 (1991), pp. 113–129.
[14] T. Hofmeister and P. Pudlak, *A Proof that Division Is Not in TC^0_2*, Research Report 447, Department of Computer Science, University of Dortmund, 1992.
[15] R. Impagliazzo, R. Paturi, and M. Saks, *Size-depth trade-offs for threshold circuits*, in Proc. 25th ACM Symposium on Theory of Computing, ACM, New York, 1993, pp. 541–550.
[16] S. Muroga, *Threshold Logic and its Applications*, Wiley-Interscience, New York, 1971.
[17] J. Myhill and W. H. Kautz, *On the size of weights required for linear-input switching functions*, IRE Trans. Electron. Comput., EC-10 (1961), pp. 288–290.
[18] P. Orponen, *Neural networks and complexity theory*, in Proc. 17th Internat. Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 629, Springer-Verlag, Berlin, 1992, pp. 50–61.
[19] I. Parberry, *A primer on the complexity theory of neural networks*, in Formal Techniques in Artificial Intelligence: A Sourcebook, Stud. Comput. Sci. Artif. Intell. 6, North–Holland, Amsterdam, 1990, pp. 217–268.
[20] I. Parberry and G. Schnitger, *Parallel computation with threshold functions*, J. Comput. System Sci., 36 (1988), pp. 278–302.
[21] N. Pippenger, *The complexity of computation by networks*, IBM J. Res. Develop., 31 (1987), pp. 235–243.
[22] A. A. Razborov, *On small depth threshold circuits*, in Proc. 3rd Scandinavian Workshop on Algorithm Theory, Lecture Notes in Comput. Sci. 621, Springer-Verlag, New York, 1992, pp. 42–52.
[23] A. A. Razborov and A. Wigderson, *$n^{\Omega(\log n)}$ lower bounds on the size of depth 3 threshold circuits with AND gates at the bottom*, Inform. Process. Lett., 45 (1993), pp. 303–307.
[24] W. L. Ruzzo, *On uniform circuit complexity*, J. Comput. System Sci., 22 (1981), pp. 365–383.
[25] K. Y. Siu and J. Bruck, *On the power of threshold circuits with small weights*, SIAM J. Discrete Math., 4 (1991), pp. 423–435.
[26] K. Y. Siu, J. Bruck, T. Kailath, and T. Hofmeister, *Depth-efficient Neural Networks for Division and Related Problems*, Tech. Report RJ 7946, IBM Research Division, 1991. IEEE Trans. Inform. Theory, 39 (1993), pp. 946–956.
[27] K. Y. Siu and V. Roychowdhury, *On optimal depth threshold circuits for multiplication and related problems*, SIAM J. Discrete Math., 7 (1994), pp. 284–292.
[28] K. Y. Siu, V. Roychowdhury, and T. Kailath, *Computing with Optimal Size Threshold Circuits*, Tech. Report, Stanford University, 1990.
[29] S. Toda, *On the computational power of PP and $\oplus P$*, in Proc. 30th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1990, 1989, pp. 514–519.
[30] J. Torán, *An oracle characterization of the counting hierarchies*, in Proc. 3rd Annual Structure in Complexity Theory Conference, IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 213–223.
[31] J. Torán, *A combinatorial technique for separating counting complexity classes*, in Proc. 16th International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 372, Springer-Verlag, New York, 1989, pp. 732–744.
[32] K. W. Wagner, *The complexity of combinatorial problems with succinct input representation*, Acta Inform., 23 (1986), pp. 325–356.
[33] I. Wegener, *Optimal lower bounds on the depth of polynomial size threshold circuits for some arithmetic functions*, Inform. Process. Lett., 46 (1993), pp. 85–87.
[34] A. C. Yao, *On ACC and threshold circuits*, in Proc. 31st IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 619–627.

SIAM J. COMPUT.
Vol. 27, No. 1, pp. 247–290, February 1998
© 1998 Society for Industrial and Applied Mathematics
011

FULLY POLYNOMIAL BYZANTINE AGREEMENT FOR $n > 3t$ PROCESSORS IN $t + 1$ ROUNDS*

JUAN A. GARAY† AND YORAM MOSES‡

Abstract. This paper presents a polynomial-time protocol for reaching Byzantine agreement in $t + 1$ rounds whenever $n > 3t$, where n is the number of processors and t is an a priori upper bound on the number of failures. This resolves an open problem presented by Pease, Shostak, and Lamport in 1980. An early-stopping variant of this protocol is also presented, reaching agreement in a number of rounds that is proportional to the number of processors that actually fail.

Key words. Byzantine agreement, consensus, distributed computing, fault tolerance, computer security

AMS subject classifications. 68M10, 68M15, 68Q22, 94C12

PII. S0097539794265232

1. Introduction.

The Byzantine agreement (BA) problem, introduced by Pease, Shostak, and Lamport in [22], is recognized as a fundamental problem in fault-tolerant distributed computing. Over the last decade or more, the problem has received a great deal of attention in the literature, and has become a testbed for a variety of models for distributed computing (see [15] for an early survey on the subject). While BA has been studied extensively over the years, the original model in which faulty processors can act in arbitrarily malicious ways has continued to withstand a complete analysis. In [22], the authors presented a protocol that solves the problem (then still referred to as the *interactive consistency problem*) in $t + 1$ rounds whenever $n > 3t$. Here n is the total number of processors and t is an a priori upper bound on the number of faulty processors possible. They also proved that no solution for $n \leq 3t$ exists, while Fischer and Lynch later showed that $t + 1$ rounds were necessary in the worst-case run of any BA protocol [16]. The protocol presented in [22], however, required the processors to send exponentially long messages and perform exponentially many steps of computation. The design of more efficient protocols is presented in [22] as an open problem, and has been the subject of many subsequent papers. This paper presents a BA protocol for $n > 3t$ that halts in $t+1$ rounds and uses only a polynomial amount of communication and computation.

This work is based on a long sequence of papers whose goal was to reduce the complexity of the protocol while maintaining good performance in terms of the number of rounds necessary for agreement. Polynomial-time BA protocols for $n > 3t$ that halt in more than $2t$ rounds (see, e.g., [11, 24]) have been known as of 1982. In 1985 Coan presented a family of BA protocols for $n > 4t$ that, for every d, halt in $t + t/d$ rounds, and require messages of size $O(n^d)$ [8]. However, Coan's protocols require ex-

* Received by the editors February 1, 1994; accepted for publication (in revised form) January 18, 1996. An early version of this work was presented at the 1993 STOC Conference and was published as *Fully polynomial Byzantine agreement in $t+1$ rounds*, in the Proceeding of the 25th Annual ACM Symposium on Theory of Computing, ACM, New York, 1993, pp. 31–41.

http://www.siam.org/journals/sicomp/27-1/26523.html

† IBM T. J. Watson Research Center,P. O. Box 704, Yorktown Heights, NY 10598 (garay@watson.ibm.com). Part of this work was completed while the author was on leave at Weizmann Institute of Science.

‡ Department of Applied Math and Computer Science, Weizmann Institute of Science, Rehovot, 76100 Israel (yoram@wisdom.weizmann.ac.il). This work was supported in part by a Helen and Milton A. Kimmelman Career Development Chair.

TABLE 1
History of BA (partial list).

Protocol		n	rounds	comm.	comp.
[PSL]	80	$3t+1$	$t+1$	$\exp(n)$	$\exp(n)$
[DFFLS,TPS]	82	$3t+1$	$2t+c$	$\text{poly}(n)$	$\text{poly}(n)$
[C1]	85	$4t+1$	$t+\frac{t}{d}$	$O(n^d)$	$\exp(n)$
[DRS,BD,C2]	86	$\Omega(t^2)$	$t+1$	$\text{poly}(n)$	$\text{poly}(n)$
[BDDS]	87	$3t+1$	$t+\frac{t}{d}$	$O(n^d)$	$O(n^d)$
[MW]	88	$6t+1$	$t+1$	$\text{poly}(n)$	$\text{poly}(n)$
[BGP1]	89	$3t+1$	$t+\frac{t}{d}$	$O(c^d)$	$O(c^d)$
[BG1]	89	$4t+1$	$t+1$	$\text{poly}(n)$	$\text{poly}(n)$
[CW]	90	$\Omega(t\log t)$	$t+1$	$\text{poly}(n)$	$\text{poly}(n)$
[BG2]	91	$(3+\epsilon)t$	$t+1$	$\text{poly}(n)\cdot O(2^{\frac{1}{\epsilon}})$	$\text{poly}(n)\cdot O(2^{\frac{1}{\epsilon}})$
This paper		$3t+1$	$t+1$	$\text{poly}(n)$	$\text{poly}(n)$

ponential local computation. Bar-Noy, Dolev, Dwork, and Strong later improved on this result, providing protocols with essentially the same round and communication behavior, but requiring only polynomial computation [2]. These protocols thus provide a tradeoff between the number of rounds required and the size of messages used, and prove that $2t$ rounds are not necessary for polynomial-time BA protocols. $(t+1)$-round polynomial-time BA protocols for $n = \Omega(t^2)$ were presented in 1986 by Dolev, Reischuk, and Strong [13]. In 1988 Moses and Waarts presented the first polynomial $(t+1)$-round protocol with linear resilience. It required only that $n > 8t$ and was later improved to handle $n > 6t$ [21]. In 1989 Berman and Garay presented a polynomial protocol for $n > 4t$ [3], which they improved in 1991 to handle $n > (3+\epsilon)t$ for any $\epsilon > 0$ [4]. At the cost of requiring more processors ($\Omega(t \log t)$), Coan and Welch developed a polynomial protocol that uses one-bit messages and asymptotically optimal total bit transfer [10]. Table 1 presents a summary of these and related results.

Our work starts out using observations and techniques developed in [2, 21, 3]. In particular, we start from the formulation by Bar-Noy et al. of the exponential protocol for $n > 3t$ in terms of a two-stage process. In the first stage, processors exchange information for $t+1$ rounds, and each processor stores the information it receives in a treelike data structure. In the second stage, each processor computes *resolved* values for each node in a bottom-up fashion, where the resolved value of a node is a function of the resolved values of its children in the tree. The resolved value of the root of the tree is the value the processor decides on. In order to obtain our result, we need to develop new techniques to handle the great powers that faulty processors have when $n > 3t$. In particular, we study the class of functions that can serve in the second stage of the algorithm for computing resolved values for the nodes. We present a new class of *admissible* resolve functions, that alternate between favoring the value 0 at one level of the tree and favoring 1 at the next level. By extending the methods for fault detection and fault masking used, we are able to present a particular admissible resolve function, which restricts the freedom of faulty processors. Given these methods, this function forces the number of processors that are detected as faulty by all nonfaulty processors to be large in runs in which the processors' trees grow exponential. This in turn makes it possible to apply a monitor voting technique along the lines of the cloture votes technique of Berman and Garay [3] to obtain a polynomial BA protocol.

A few comments are in order regarding the related *randomized* Byzantine agreement problem. Rabin showed that a shared coin can be used to attain randomized

agreement in a constant number of rounds [23]. Later, Feldman and Micali presented an algorithm for constructing such a coin, thereby providing an optimal protocol for this problem [17]. In the protocol of [17], however, the decision property was guaranteed only with probability 1, and the protocol has runs in which no processor ever reaches a decision. Moreover, there is no finite bound K, such that the processors that do decide, do so in K rounds. Indeed, there is no obvious way to transform such a probabilistic protocol into a polynomial protocol in which nonfaulty processors never violate the conditions of Byzantine agreement (agreement and validity), and yet they all decide in fewer than $t+1$ rounds with probability 1. A very close approximation of this was obtained by Goldreich and Petrank [19]. They presented a probabilistic Byzantine agreement protocol with a constant expected number of rounds that is guaranteed to always halt in $t + O(\log t)$. Their protocol is based on a sequential combination of a probabilistic protocol with a deterministic $(t+1)$-round protocol. Before our paper, their scheme, when applied to the case of $n > 3t$, would yield a protocol with an exponential worst-case complexity in terms of both computation and communication. Using our protocols in their scheme, it is now possible to obtain such a probabilistic protocol for $n > 3t$ with polynomial-time worst-case complexity. Using different techniques, Zamsky has recently improved to $t+1$ the worst-case running time of [19], while maintaining linear, but nonoptimal, resiliency [27].

The remainder of the paper is organized as follows. In section 2 we describe the model and formally define the problem. In section 3 we review EIG, the exponential information gathering protocol of Bar-Noy et al. In section 4 we review the techniques that made polynomial-time protocols possible for $n > 4t$. Section 5 is the first step towards a polynomial solution for $n > 3t$. We introduce a general, radically different class of resolve functions. This class of functions is then used in section 6 as the basis of a protocol for a slightly generalized version of Byzantine agreement. In section 7 we present the concrete function of our choice, while in section 8 we present `Sliding-flip`, the polynomial-time protocol for $n > 3t$. Finally, in section 9 we modify the protocol so as to terminate in $\min\{t+1, f+3\}$ rounds, where $f < t$ is the actual number of failures that occur in the run.

2. Preliminary definitions. For ease of exposition of our protocols we shall be concentrating on the following variant of Byzantine agreement, sometimes also called the *consensus* problem, which is defined as follows: given are n processors, at most t of which might be faulty. Each processor i has an initial value $v_i \in \{0,1\}$. Required is a protocol with the following properties:

(i) Decision: Every nonfaulty processor i eventually irreversibly "decides" on a value $d_i \in \{0,1\}$.

(ii) Agreement: The nonfaulty processors all decide on the same value.

(iii) Validity: If the initial values v_i of all nonfaulty processors are identical, then $d_i = v_i$ for all nonfaulty processors i.

In the other common variant of Byzantine agreement there is a distinguished leader with a single initial value v. The agreement and decision conditions remain the same, while the validity condition requires that if the leader is nonfaulty, all nonfaulty processors should decide on the leader's value v. Protocols for both variants are practically identical, and everything we say here applies to the single leader case with only minor modifications.

Throughout the paper we use t to denote an upper bound on the number of faulty processors. We assume the standard model for Byzantine agreement, in which processors may fail in arbitrarily malicious ways (see, for example, [16]). Each proces-

sor can communicate directly with every other processor via a reliable point-to-point channel. Finally, the processors are synchronous and their communication proceeds in synchronous rounds. In a round, a processor can send one message to each of the other processors, and all messages are received in the round in which they are sent.

3. Exponential information gathering protocols. We start by describing the exponential protocol due to Bar-Noy et al. [2] (Exponential Information Gathering (EIG)), which is closely related to the original protocol of Pease, Shostak, and Lamport [22]. Our final protocol will be obtained by a sequence of transformations to this protocol, based on distinct observations. Our description of the EIG protocol below is essentially taken from Bar-Noy et al. [2].

In the first round of the EIG protocol for Byzantine agreement, each processor broadcasts its initial value v_i to all other processors. In each of the following t rounds, every processor broadcasts all of the information it received in the latest round. At the end of $t+1$ rounds each processor computes a decision value based on the information it has gathered, decides on this value, and halts.

We now describe the protocol in greater detail. Each processor incrementally constructs a tree-based data structure which we will call an EIG *tree*. We consider the root of the tree to be a node of *depth* 0, and inductively define the depth of a node to be greater by one than the depth of its parent. We will only be interested in EIG trees of depth at most $t+1$. In the EIG tree, a node of depth r has $n-r$ children. Thus, in particular, the root has n children. The edges of the EIG tree are labelled with processor names as follows. The outgoing edges of the root are labelled $1, \ldots, n$, respectively. A node of depth $r \geq 1$ has an edge labelled i for every processor name i that does not appear on the path leading from the root to the node. Notice that with this definition no label appears twice on a path from the root to a leaf. It follows that the sequence of labels on the path from the root to a given node uniquely determines this node. Moreover, there is a 1–1 correspondence between strings σ of up to $t+1$ distinct processor names and the nodes in an EIG tree of depth $t+1$. We will thus regard such a string as the *name* of the corresponding node, and refer to nodes by such names. The root is named by λ, which stands for the empty string. Notice that the length of the string σ, which we shall denote by $|\sigma|$, coincides with the depth of σ.

We shall ultimately associate two values with each node σ in a processor i's tree: a *stored value*, denoted by $\mathtt{tree}(\sigma)$, and a *resolved value* denoted by $\mathtt{res}(\sigma)$. When we need to specify the particular processor i in whose tree these values appear, we denote them by $\mathtt{tree}_i(\sigma)$ and $\mathtt{res}_i(\sigma)$, respectively. The stored values are assigned during the $t+1$ rounds of information exchange between the processors, while the resolved values are computed at the end, in order to determine the decision value. The manner in which these values are arrived at is described below. A node σj is said to *correspond* to the processor j whose name labels the edge leading to the node from its parent σ. We call a node of the EIG tree *correct* if it corresponds to a nonfaulty processor.

Each processor i initially stores its initial value v_i in $\mathtt{tree}_i(\lambda)$—at the root of its EIG tree. In the first round of the EIG protocol a processor will send the value stored in its root to all n processors (including itself, although of course this message need not actually be sent). For every processor j, the value that j sends to i in the first round will be stored in the node $\langle j \rangle$ of $\mathtt{tree}_i$ (a default value of 0 will be stored in $\mathtt{tree}_i(\langle j \rangle)$ in case processor j does not receive a legitimate message with a value from j). In each subsequent round every processor sends to all other processors the level of its tree most recently filled in. (Note that a faulty processor may of course send

different values than the ones it should send, and also may send conflicting messages to different processors in the same round.) The messages received are broken up and used to form a new level in the processor's tree as follows: If j's message reports the value v for the node σ, and j does not appear in σ, then the value v will be stored at the node σj. Again, we store a default value of 0 in σj in case j did not report a legitimate value for σ (that is, if j did not send a message that is in the syntax appropriate for messages sent in this round). Intuitively, the node σj in $\texttt{tree}_i$ stores the value that j claims in its message to i to have stored in $\texttt{tree}_j(\sigma)$. Notice that the single message received by i from j in a given round reports on the values of many nodes in j's tree, and will be used to update many different nodes in $\texttt{tree}_i$.

We consider a node to be *created* in the round in which a value is stored in the node. In the first stage of the EIG protocol, information is gathered for $t+1$ rounds, until all nodes of depth up to $t+1$ are created. At that point each processor computes a value for the tree by applying the recursive computation of $\texttt{res}_i(\cdot)$ to the root λ. The processor then "decides" on the value of $\texttt{res}(\lambda)$, and halts. For the purpose of the current section and the next one, the particular function used for $\texttt{res}(\cdot)$ is *recursive majority voting*, defined as follows:

$$\texttt{res}_i(\sigma) = \begin{cases} \texttt{tree}_i(\sigma) & \text{if } \sigma \text{ is a leaf;} \\ 1 & \text{if majority of } \texttt{res}_i(\sigma j) \text{ are 1;} \\ 0 & \text{otherwise.} \end{cases}$$

Thus, $\texttt{res}_i(\sigma)$ computes the value of the recursive majority of the descendants of the node σ in $\texttt{tree}_i$. This completes the description of the EIG protocol. Bar-Noy et al. prove the following theorem.

THEOREM 3.1 (see Bar-Noy et al. [2]). *Protocol* EIG *solves the Byzantine agreement problem for* $n > 3t$.

4. Polynomial protocols for $n > 4t$. While the EIG protocol is a correct $(t+1)$-round BA protocol for $n > 3t$, it has a major drawback in requiring messages and local memory of exponential size. In fact, *every* run of this protocol is exponential. Nevertheless, the clean structure provided by the EIG tree will allow us to apply a sequence of transformations to this protocol, and to finally arrive at a polynomial protocol. In this section we shall present the basic observations that lead to a polynomial protocol in the case of $n > 4t$. In section 5 we shall highlight the problems and discuss the modifications required in the case of $n > 3t$. This will require a careful analysis and new techniques. In particular, we shall introduce a new class of resolve functions, which are instrumental in achieving a polynomial protocol for $n > 3t$.

4.1. Predicting resolved values. The first step is based on an observation due to Moses and Waarts in [21]. A processor i can often determine early on what the value of $\texttt{res}_i(\sigma)$ in the EIG protocol will be. This is commonly called *prediction*. Roughly speaking, once i is able to predict the resolved value of σ, it can stop gathering information about the whole subtree of $\texttt{tree}_i$ rooted at σ. One basis for prediction is the following lemma that is used in the proof of Theorem 3.1.

LEMMA 4.1 (see Bar-Noy et al. [2]). *Let $n > 3t$, let i and j be nonfaulty processors, and let σ be a node of depth $|\sigma| \leq t$ such that j does not appear in σ. Then at the end of round $t+1$ we have $\texttt{res}_i(\sigma j) = \texttt{tree}_i(\sigma j) = \texttt{tree}_j(\sigma)$.*

As discussed in [21] if, for example, *majority* $+t-1$ children of σ store the same value v in $\texttt{tree}_i$, then i knows that at least majority of children of σ are correct and store v. Hence, by Lemma 4.1 we get that the majority of $\texttt{res}_i(\sigma j)$ values will be v. The definition of the resolve function now implies that we will end up with

$\texttt{res}_i(\sigma) = v$. Following [21], we will say in this case that the value of σ is *fixed to* v *for* i once σ's children in $\texttt{tree}_i$ have been created. Another case in which the resolved value of a node can be predicted is when a majority of its children become fixed to the same value. We also define a node σ to be *closed* in $\texttt{tree}_i$ at the end of round r if either σ or one of its ancestors is fixed in $\texttt{tree}_i$ at that time. The properties of prediction in the case of $n > 4t$ are described in the following two statements.

LEMMA 4.2 (see Moses and Waarts [21]). *Assume $n > 4t$, let σ be a correct node with $|\sigma| \leq t$, and let i be a nonfaulty processor. Then σ is closed in $\texttt{tree}_i$ by the end of round $|\sigma| + 1$.*

COROLLARY 4.3 (see Moses and Waarts [21]). *Assume $n > 4t$ and let i and j be nonfaulty processors. In the* EIG *protocol, if σ is fixed for i at the end of round r, then σ is closed in $\texttt{tree}_j$ at the end of round $r + 1$ at the latest.*

An immediate implication of Corollary 4.3 is that a nonfaulty processor need report on descendants of a node for at most one round after it can determine that the node is fixed to some value. No further information it can send about the subtree of that node will help anyone determine its final resolved value. In particular, once the root becomes fixed in $\texttt{tree}_i$, processor i knows the decision value and needs to send information for at most one more round. Moses and Waarts [21] use this to devise an early-stopping BA protocol for $n > 4t$. This protocol is still exponential, although it is significantly more efficient than the one that constructs the full EIG tree. Moreover, there are many cases in which the tree is polynomial in size. In fact, in the common case in which no failures arise, the protocol ends after two rounds, and the amount of communication sent by a processor is $O(n)$ in the case of single-source BA and $O(n^2)$ in the consensus case.

Lemma 4.2 provides some insight into the relationship between the size of the EIG trees (and hence the complexity of computation and communication) in a given run of the early-stopping BA algorithm and the behavior of the faulty processors during that run. Let us call a node σ *corrupted in* $\texttt{tree}_i$ if σ is not closed in $\texttt{tree}_i$ by the end of round $|\sigma| + 1$. Clearly, a noncorrupted node can have at most n children in the tree. In addition, it is not hard to check that a corrupted node can account for the need to store its grandchildren, but not for any later descendants. The total size of an EIG tree thus becomes roughly $O(n^2 C)$, where C denotes the number of corrupted nodes in the tree. It follows that if we are able to reduce the number of corrupted nodes, then the trees will shrink, and hence also the communication. (Indeed, the protocol of [21] managed, for $n > 8t$, to ensure that no processor is able to corrupt more than one node overall, and thus in all trees $C \leq t$.) A central tool in reducing the number of corrupted nodes turns out to be the detection and masking of faulty processors, which we now turn to discuss.

4.2. Detecting and masking failures. Lemma 4.2 implies that if σj is corrupted in $\texttt{tree}_i$, then processor i can detect that j is faulty immediately after it receives values for σj's children. Indeed, we shall see later on some other instances in which a processor i can detect another processor as faulty. Detecting failures is useful because of the following observation, which was first made by Dolev, Reischuk, and Strong [13] and later used in various ways in [8, 2, 21, 3] and others. Since faulty processors can send arbitrary messages, once we detect a processor as being faulty, we can act as if it sends us particular messages that may be to our advantage, ignoring what it actually sends us. This is called *fault masking*. One particularly simple form of fault masking is to regard a processor detected as faulty to send us a fixed value (e.g., 0) for all nodes of interest. As we shall see, this can help reduce the number

of corrupted nodes. Once all nonfaulty processors mask a given processor z, we say that z is *disabled*. Notice, in particular, that once z is disabled, nodes of the form σz can no longer be corrupted. They become fixed to the value being used for masking at the end of round $|\sigma z| + 1$.

Corollary 4.3 implies that unless σj is corrupted in all of the correct processors' trees, it will become closed (in all trees) at the end of round $|\sigma j| + 2$. It follows that in order for the subtree of σj to grow to a substantial size in some processor's tree, the node σj must be corrupted in *all* nonfaulty processors' trees. We will call a node *universally corrupted* if it is corrupted in all nonfaulty processors' trees. Another consequence of Corollary 4.3 is that the final size of an EIG tree is polynomial if and only if the number of universally corrupted nodes in the tree is polynomial. This follows from the fact that if the subtree of a universally corrupted node has more than n^3 nodes, at least one of the node's children must also be universally corrupted. If a node σj is universally corrupted, then everyone detects j as faulty at the end of round $|\sigma j| + 1$, and j becomes disabled from that point on. It cannot corrupt nodes in later rounds. It follows that when $n > 4t$, fault masking allows a faulty processor to universally corrupt nodes only during a single round. Moreover, Lemma 4.2 implies that if neither the node σj nor any of its ancestors is fixed in $\texttt{tree}_i$ by the end of round $|\sigma j|$, then all processor names appearing in the string σ denote faulty processors. This implies that once we employ early stopping and fault masking in the case of $n > 4t$, at least $3t$ of the children of any node of interest are nonfaulty. Moreover, at most $t - |\sigma|$ children of a node σj can be faulty. As a result, if processor i finds more than $t - |\sigma|$ values among the children of σj that differ from $\texttt{tree}_i(\sigma j)$, it can detect that j is faulty.

If no faulty processor universally corrupts a node in a given round r, then all nonfaulty processors decide by the end of round $r+1$. If only one processor universally corrupts nodes in each round (and we employ early stopping and fault masking), then the total size of the tree is polynomial. Following [3], we shall call ESFM (early stopping with fault masking) the protocol resulting from the combined application of the prediction and fault masking techniques to EIG. The ESFMprotocol significantly reduces the size of the EIG tree processors construct.[1] However, as discussed in [21], the trees may still grow exponential in general because (i) more than one processor can corrupt nodes in a given round, and (ii) a processor can corrupt many nodes in a given round. The final observation that will yield a polynomial protocol, at least in the case of $n > 4t$, is that when many processors universally corrupt nodes in a given round, it is possible for the nonfaulty processors to detect that something "fishy" is going on, and to decide to stop. How to do so is the subject of our next subsection.

4.3. Monitor voting. An important new idea was introduced by Berman and Garay in [3]. They proposed to have each processor observe the size of its tree, and when the size exceeds a certain threshold, to "vote" to stop the whole agreement procedure. Thus, roughly speaking, they started a new instance of Byzantine agreement in every round, in which a processor's initial value reflects whether or not the processor is in favor of stopping the process (and deciding on a default value). In order for any processor's tree to become exponential, all processors' trees must grow beyond any polynomial threshold. Should this happen, all nonfaulty processors would vote in favor of stopping and deciding on the default, and this instance of the agree-

[1] Using both techniques—prediction and fault masking—simultaneously reduces the tree from size $O(n^t)$ to $O(c^t)$ (see [21]). In [4] a more sophisticated masking technique yields a similar result for $n > 3t$.

ment protocol would stop in two rounds. In the [3] protocol, once such an agreement is attained, every nonfaulty processor decides on the default and halts, terminating its participation in all Byzantine agreement instances underway.[2] We think of the agreement instances initiated in every round of the [3] protocol as processes that *monitor* the run, ensuring that trees do not grow too much. Indeed, Berman and Garay showed that when performed in a careful way, the monitor voting technique yields the following theorem.

THEOREM 4.4 (see Berman and Garay [3]). *By applying monitor voting to the* ESFM *tree, it is possible to obtain a* $(t+1)$*-round polynomial-time BA algorithm for* $n > 4t$.

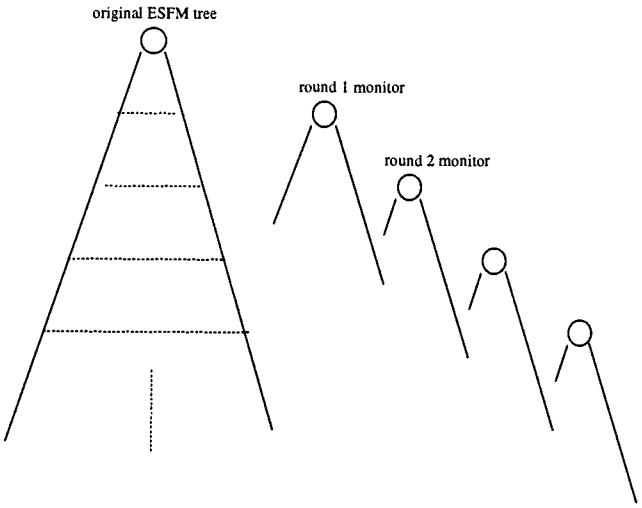


FIG. 1. *The monitor voting technique.*

The technique is illustrated in Figure 1. Monitor voting introduces a number of new subtleties.

(i) Once there are a number of agreement processes underway, we must be careful to ensure that processors use the individual agreements' values in a consistent way when determining their ultimate decision value. If, for example, one processor may decide 0 because its original information gathering tree stopped with value 0, while the other decides 1 because one of the monitors stopped with value 1, we are in trouble.

(ii) As monitor processes are initiated in different rounds of the execution, we need to take action to ensure that they all halt by time $t+1$. Otherwise, we may run into consistency problems as described above, when we are forced to decide at time $t+1$.

(iii) It may seem desirable to exclude faulty processors from participating in a monitor, in order to ensure the monitor halts in time. However, deciding that a given processor is faulty is as hard as Byzantine agreement itself. Techniques of [12] can be used to show that we cannot exclude processors "on the fly," as a result of failures being detected.

[2] This technique was called *cloture votes* by the authors, since it resembles a procedure in the U.S. Senate in which a vote can be called on whether the discussion on a given topic has been going on for too long.

The main tool that can be used to limit the damage caused by faulty processors is to use information about failures that a processor has gained in one agreement process to mask faulty processors in all agreement processes. It is crucial, however, to ensure that sufficiently many processors are *disabled* before a processor can vote in favor of stopping in a monitor agreement process. In fact, we shall use the following rule for determining the initial value of processor i in monitor process $\mathtt{M}^r$, which is the monitor agreement process initiated in round r:

Monitor-vote: Processor i will vote 1 on $\mathtt{M}^r$ if it has detected that $r-1$ faulty processors have corrupted nodes and are disabled by the end of round $r-1$; it will vote 0 on $\mathtt{M}^r$ otherwise.

In the case of $n > 4t$, it is possible to show that for the information gathering trees to grow large, the number of disabled processors must grow sufficiently fast to cause every nonfaulty processor to eventually vote 1 using this rule. This is the basis of the solution for $n > 4t$ given in [3]. Extending this idea to $n > 3t$, however, seems to be problematic, because the faulty processors have greater powers to corrupt nodes when $4t \geq n > 3t$.

5. Deriving a polynomial protocol for $n > 3t$. Our goal in this paper is to describe a polynomial BA protocol for $n > 3t$ that is guaranteed to terminate in $t+1$ rounds. Intuitively, the protocol follows the monitor voting approach presented in the previous section. In the case of $n > 3t$, however, most of the properties used when $n > 4t$ no longer hold. First of all, using the resolve function defined in section 4, the analogues of Lemma 4.2 and Corollary 4.3 fail, i.e., it is possible to show that when $n < 4t$ the majority function does not guarantee quick prediction of the value $\mathtt{res}(\sigma)$ even for correct nodes σ. (Indeed, we know of no constant number of generations of descendants for which `res` guarantees prediction of correct nodes.) In [25] Waarts presented prediction rules for a resolve function introduced by Bar-Noy et al. [2], and showed that they guarantee prediction in two rounds. In the case of $n > 4t$ we had the property that exponential size of trees requires many faulty processors to expose themselves (and hence become disabled) early in the run. Intuitively, we say that some faulty processors must be "wasted" for the adversary to be able to cause trees to grow exponential. This was crucial for the monitor voting scheme to succeed in keeping the trees polynomial. As we shall see, the fact that it sometimes takes nodes two rounds to fix when $n > 3t$ implies that there are often cases in which the tree can grow without requiring such waste to occur. The function used by Waarts in [25] is ternary, and it occasionally provides an "undecided" value for an internal node. This introduces enough "slack" into the information conveyed by the resolve function, that we were unable to use it successfully in the case of $n > 3t$. It seems possible for the adversary to cause the trees to grow exponential when that resolve function is used, without the number of disabled processors ever becoming large enough to be "caught" via monitor voting. A major focus of this paper is on deriving a resolve function and corresponding prediction and fault detection rules that will guarantee that exponential growth of the agreement trees will require sufficient waste to enable monitor voting to detect the problem and halt.

In the rest of this section we introduce a radically different class of resolve functions. This class of functions is then used in section 6 as the basis of a protocol for a slightly generalized version of Byzantine agreement. We end up chosing the precise function from this class in such a way that it guarantees that for the size of the EIG trees to grow beyond a polynomial bound, faulty processors must be disabled at a rate that is greater than one per round. This is the key point that enables us to perform

monitor voting for the case of $n > 3t$ in the spirit of the monitor voting we performed for $n > 4t$.

5.1. A general class of resolve functions. Let us denote the set $\{0, \ldots, j\}$ by $[j]$. For every function $F : [t] \to [n]$, we define a resolve function $\texttt{res}^F$ as follows:

$$\texttt{res}^F(\sigma) = \begin{cases} \texttt{tree}(\sigma) & \text{if } \sigma \text{ is a leaf;} \\ 1 & \text{if at least } F(|\sigma|) \text{ of } \texttt{res}^F(\sigma j) \text{ are 1;} \\ 0 & \text{otherwise.} \end{cases}$$

Notice that $\texttt{res}^F(\sigma) = 0$ if and only if $\#\{j : \texttt{res}^F(\sigma j) = 0\} \geq n - |\sigma| + 1 - F(|\sigma|)$. Moreover, notice that the original res function of [2] is simply $\texttt{res}^M$, where M is the majority function. Let us now turn to a straightforward observation regarding functions of this type. Let a *cut* through the EIG tree be a set C of nodes that intersects every path from the root to a leaf exactly once. One important property of resolve functions is that the values of a resolve function on the nodes of a cut uniquely determine its values on all nodes above it.

LEMMA 5.1. *Let C be a cut of the* EIG *tree, and let $F : [t] \to [n]$. Then the values of $\texttt{res}^F$ on the nodes of C uniquely determine the values of $\texttt{res}^F(\sigma)$, for all nodes σ that are ancestors of nodes in C.*

Proof. Notice that because C is a cut, if a node σ is an ancestor of a node in C, then every one of σ's children is either in C or an ancestor of a node in C. The proof is by induction on the height of σ above C. Assume that σ is an ancestor of a node in C and that the values of $\texttt{res}^F$ on the nodes of C uniquely determine the values of all of σ's children σj. Since $\texttt{res}^F(\sigma)$ depends only on the values of $\texttt{res}^F$ on σ's children, and their values are uniquely determined by the values of $\texttt{res}^F$ on C, we are done. □

This lemma has an immediate corollary that we shall find most useful in the sequel.

COROLLARY 5.2. *Let $F : [t] \to [n]$. If, for all nodes σ of some cut C it is the case that $\texttt{res}^F(\sigma)$ has the same value in all correct processors' trees, then the values of $\texttt{res}^F$ on every node above C are the same in all trees. In particular, this applies to $\texttt{res}^F(\lambda)$, the resolved value for the root.*

We are now in a position to identify a large class of functions F that can serve as resolve functions for Byzantine agreement protocols. We say that F is a *sound* resolve function for Byzantine agreement if applying $\texttt{res}^F$ to the complete EIG trees of nonfaulty processors is guaranteed to fulfill the requirements of BA. (Notice that decision is trivial in this case, since it is assumed that a processor decides on $\texttt{res}^F(\lambda)$; thus, the claim actually concerns the agreement and validity properties).

LEMMA 5.3. *If $F : [t] \to [n]$ satisfies $t + 1 \leq F(r) \leq n - t - r$, then $\texttt{res}^F$ is a sound resolve function for Byzantine agreement for $n > 3t$.*

Proof. The proof proceeds along the lines of the proof of $\texttt{res}^M$ (the majority resolve function) from [2]. First notice that we are only interested in applying this rule for depths r satisfying $r \leq t$. In this case, if $n > 3t$ then indeed $n - t - r \geq t + 1$, so such functions $F(r)$ exist.

Let $h_L(\sigma) = t + 1 - |\sigma|$ be the height of the node σ above the leaves. First, we show by induction on $h_L(\sigma)$ that every correct node σ is resolved to $\texttt{tree}(\sigma)$. For a leaf σ ($h_L = 0$) this is immediate from the definition of $\texttt{res}^F$. Assume the claim is true for all children of σ, and that σ is a correct node with $h_L(\sigma) \geq 1$. It follows that at least $n - t - |\sigma|$ of σ's children j are correct. Moreover, since σ is correct, all correct processors $j \notin \sigma$ receive and echo an identical value for σ, so that $\texttt{tree}(\sigma j) = \texttt{tree}(\sigma)$.

By the inductive hypothesis, we thus have $\#\{j : \mathtt{res}^F(\sigma j) = \mathtt{tree}(\sigma)\} \geq n - t - |\sigma|$. The claim is obtained by observing that $n - t - |\sigma| \geq F(|\sigma|)$, which covers the case $\mathtt{tree}(\sigma) = 1$, and $n - t - |\sigma| = n - |\sigma| + 1 - (t + 1) \geq n - |\sigma| + 1 - F(|\sigma|)$, which covers the case of $\mathtt{tree}(\sigma) = 0$. This completes the inductive argument.

To complete the proof, notice that a tree of depth $t + 1$ has at least one correct node on every path from the root to the leaves. It follows that there is a cut C through the tree consisting of correct nodes only. For correct nodes σ (and, in particular, all nodes $\sigma \in C$) we have just shown that $\mathtt{res}^F(\sigma)$ is the same in all correct processors' trees. It follows by Corollary 5.2 that $\mathtt{res}^F(\lambda)$ is the same in all trees, and we are done. □

Notice that Lemma 5.3 allows a wide range of functions $F(r)$, including the natural function M (majority). Our quest to lower the complexity of the algorithm requires that we do not expand the whole EIG tree. Rather, we wish to use information we gather incrementally in order to be able to predict the resolved values of nodes, and thereby hopefully be able to avoid expanding their subtrees. Once every processor is able to predict the resolved values of nodes on a cut of the tree, we shall be done. Indeed, as shown in [21], when $n > 4t$ the function $\mathtt{res}^M$ guarantees that the resolved values of correct nodes can always be predicted once values of their children are stored. Unfortunately, when $n < 4t$ majority does not guarantee quick prediction of the value of $\mathtt{res}^M(\sigma)$ even for correct nodes σ. We shall thus seek other functions that will allow effective prediction.

We shall be most interested in a particular type of resolve function satisfying the conditions of Lemma 5.3 which, roughly speaking, will alternate between favoring the value 0 and favoring the value 1 in consecutive rounds.[3] (Intuitively, a favored value is one that requires a minimal amount of support in a given round in order to force a parent to be resolved to this value, while the opposite value requires a substantial amount of support.) A major consequence of this "flipping" function will be that the resolved values of correct nodes can be predicted in two rounds at the latest. More specifically, the alternating behavior will be controlled by the *parity* of the round. We let $\mathtt{par}(r)$ denote the parity of r. Thus, $\mathtt{par}(0) = 0$, $\mathtt{par}(1) = 1$, $\mathtt{par}(2) = 0$, etc.

Another important property of our function is that it will only be defined for an appropriate portion of the EIG tree, and not for the complete tree. More specifically, let us define a node σ to be *righteous* if σ is a correct node, and all of its ancestors are faulty. (In other words, a righteous node is the first correct node on some path from the root to a leaf.) Clearly, the identity of the faulty processors in a given run completely determines which nodes of the tree are righteous. An important property of the EIG tree is that given any choice of up to t faulty processors, the set of righteous nodes forms a cut in the EIG tree. Given a set B of up to t faulty processors, we define the *righteous subtree* of the EIG tree (with respect to B) to be the tree whose leaves are the righteous nodes, and whose internal nodes consist of the ancestors of righteous nodes. Clearly, different choices for B yield different righteous subtrees. (In the sequel, the set B will be implicit, and will consist of the faulty processors in the run under discussion.) Our goal will be to devise a mechanism by which, roughly speaking, righteous nodes will be resolved to their stored values in all nonfaulty processors' trees. Since the righteous nodes are, in particular, correct nodes, this means that they will be resolved to the same value in all trees. Once

[3] A protocol that similarly alternates between favoring 0 and favoring 1, albeit in a different setting, has previously been used by Attiya et al. in [29]. In our setting, alternation has been used for fault masking by Berman and Garay in [4].

righteous nodes resolve to the same values, we can use them in the role of leaves in the definition of a new resolve function. We thus define a resolve function $\texttt{right}^F$ on nodes σ of the righteous subtree as follows:

$$\texttt{right}_i^F(\sigma) = \begin{cases} \texttt{tree}_i(\sigma) & \text{if } \sigma \text{ is righteous;} \\ \texttt{par}(|\sigma|) & \text{if } \texttt{right}_i^F(\sigma j) = \texttt{par}(|\sigma|) \text{ for at least } F(|\sigma|) \text{ nodes } \sigma j\text{; and} \\ 1 - \texttt{par}(|\sigma|) & \text{otherwise.} \end{cases}$$

Notice that $\texttt{right}_i^F(\sigma)$ is undefined for nodes σ outside the righteous subtree. Since, for a righteous node σ, we are guaranteed that $\texttt{tree}_i(\sigma) = \texttt{tree}_j(\sigma)$ holds for all (nonfaulty) i and j, it is the case that $\texttt{right}_i^F = \texttt{right}_j^F$. Since the righteous nodes form a cut in the tree, it follows by Lemma 5.1 that $\texttt{right}_i^F$ is independent of i. Therefore, we shall henceforth drop the subscript and refer to this function by $\texttt{right}^F$. We now have the following proposition.

PROPOSITION 5.4. *$\texttt{right}^F$ satisfies the agreement and validity conditions of Byzantine agreement for every resolve function $F : [t] \to [n]$.*

The definition of $\texttt{right}^F$ seems to have a drawback. Processors are, in general, unable to tell a righteous node from a nonrighteous one. So how can $\texttt{right}^F$ be used by the processors? Intuitively, our plan is to present "fixing" rules that would predict the $\texttt{right}^F$ value of a node, provided the node is in the righteous subtree. For nodes outside the righteous subtree the $\texttt{right}^F$ value is undefined, and we give no guarantee on what values our fixing rules may give. This will not cause problems because our rules will have the property that for every righteous node σ, either σ or one of its ancestors will be fixed by the time values for σ's grandchildren are stored in the tree (i.e., by the end of round $\min(t+1, |\sigma|+2)$. Thus, our scheme will guarantee that all nodes of the righteous subtree, including the root, become fixed in all trees by the end of round $t+1$, if not earlier. Moreover, the nodes of the righteous subtree, when fixed to a value, will be fixed to their $\texttt{right}^F$ value. We shall now turn to formalize this intuition.

6. Δ-agreement. The monitor processes we talk about are agreement protocols that closely resemble ordinary Byzantine agreement, except for the following differences. (a) A monitor process is initiated in a state in which each nonfaulty processor i has a set $\mathcal{F}_i$ of processors that i has already detected as faulty, and is masking throughout the monitor. (b) The set of faulty processors used by processor i in each of the monitors is obtained based on fault detection performed by i in *all* active agreement processes. (c) Finally, we will associate with a monitor $\texttt{M}$ a parameter $\Delta < t$ which, roughly speaking, is a lower bound on the number of initially disabled faulty processors. This gives rise to a slight generalization of Byzantine agreement, that we shall call Δ-agreement.

We define Δ-agreement more formally as follows. As in Byzantine agreement, in an instance of Δ-agreement, each nonfaulty processor i starts out with an initial value $v_i \in \{0,1\}$. In addition, every nonfaulty processor i starts out with a set $\mathcal{F}_i$ of faulty processes. If all nonfaulty processors start out with an initial vote of 0, the parameter Δ plays no role. If, however, at least one nonfaulty processor votes 1, then at least Δ faulty processors are initially disabled. Let $\mathcal{D} \stackrel{\text{def}}{=} \bigcap_i \mathcal{F}_i$ denote the set of initially disabled processors. In Δ-agreement, we are guaranteed that if at least one nonfaulty processor votes 1, then $\#\mathcal{D} \geq \Delta$. (Thus, if all nonfaulty processors vote 0, no guarantee about the size of $\mathcal{D}$ is given.) Strictly speaking, an instance

of Δ-agreement has two additional parameters, n and t, where as usual n is the total number of processors, and t is an upper bound on the total number of faulty processors (including the ones in $\mathcal{D}$ and in the $\mathcal{F}_i$'s). Rather than working explicitly with (n, t, Δ)-agreement, we shall continue to talk about Δ-agreement, keeping n and t implicit and assuming that n and t satisfy $n \geq 3t + 1$, while $\Delta < t$. The decision, agreement, and validity requirements are as in the case of Byzantine agreement defined in section 2. Notice that the standard variant of Byzantine agreement that we have been considering can be viewed as an instance of Δ-agreement, where $\Delta = 0$ and $\mathcal{F}_i = \emptyset$ for every nonfaulty processor i.

6.1. Basic structure of the Δ-EIG protocol. Our purpose in this section will be to describe a protocol for Δ-agreement, which we shall call the Δ-EIG protocol. In later sections we shall discuss how to combine a number of these protocols via monitor voting to obtain an efficient solution to Byzantine agreement.

Our Δ-EIG protocol for a single instance of Δ-agreement will be based on a number of components. We now discuss some of them. First of all, the Δ-EIG protocol will operate on an EIG tree of depth $t + 1 - \Delta$ (as opposed to depth $t + 1$ in the standard EIG protocol). This will be essential, as we want to be able to complete a run of this protocol within $t+1-\Delta$ rounds. In addition, we shall have every processor i maintain a set of processors it has detected as faulty. Let $\mathcal{F}_i(r)$ denote the set of faulty processors detected by i in the first r rounds. Thus, in particular, $\mathcal{F}_i(0) = \mathcal{F}_i$ and the sets $\mathcal{F}_i(r)$ grow monotonically over time (i.e., $\mathcal{F}_i(r + 1) \supseteq \mathcal{F}_i(r)$). These sets will be used by the processors both for masking values in their own trees and for reporting on masked nodes. Rather than processor i sending in round $r + 1$ separate reports of masked values for each node τz corresponding to processors $z \in \mathcal{F}_i(r)$, we shall have i send a report of the form $\texttt{mask}(i, z)$ in the first round following the one in which it detects z to be faulty. All processors that receive this report will, from then on, act as if i actually sends separate masked reports for such nodes. In fact, we shall abuse the language slightly, and consider a processor i that issues a $\texttt{mask}(i, z)$ report in round r as if it "reports" masked values for all nodes σz with $|\sigma z| \geq r$. A processor will keep track of $\texttt{mask}(i, z)$ reports it receives and will store masked values in nodes accordingly on i's behalf, as specified below. In addition to saving in communication, this will allow a processor to detect failures based on reports its receives, and will, later on, make it easier for i to estimate the number of disabled processors.

Formally, processors will send messages according to the following rule.

- Sending: In a given round $r + 1$, a processor i sends all other processors a message consisting of two components. In the first component, the message contains reports $\texttt{mask}(i, z)$ on the processors $z \in \mathcal{F}_i(r) \setminus \mathcal{F}_i(r - 1)$ that i has just discovered as faulty. For completeness, we formally define $\mathcal{F}_i(-1) = \emptyset$, so that in the first round i reports that it is masking the processors in $\mathcal{F}_i(0) = \mathcal{F}_i$. The second component of the message consists of pairs $\langle \sigma j; v \rangle$, where $v = \texttt{tree}_i(\sigma j)$, for all nodes σj of depth r such that $j \notin \mathcal{F}_i(r)$.

Upon receiving the messages sent to it in round r, processor i will record and mask values as follows.

- Recording and masking:

1. Processor i appends every $\texttt{mask}(z, j)$ report it receives in round r to a list of mask reports that it maintains;

2. Processor i records values in $\texttt{tree}_i$ according to the following:

a. If $j \notin \mathcal{F}_i(r-1)$, then $\texttt{tree}_i(\tau j)$, for a node τj of depth r, is the value reported by j for τ in round r. In particular, if $\tau = \sigma z$ for some z such that i has received a

$\texttt{mask}(j, z)$ report from j in one of the first r rounds, then $\texttt{tree}_i(\tau j) = \texttt{par}(|\tau j|)$.

b. If $z \in \mathcal{F}_i(r-1)$ and τz is a node of depth r, then $\texttt{tree}_i(\tau z) = \texttt{par}(|\tau|)$.

In particular, this means that values of nodes corresponding to initially detected failures are always masked.

The set $\mathcal{F}_i$ is updated in the following manner.

- Fault detection: $\mathcal{F}_i(r)$ is obtained by adding to $\mathcal{F}_i(r-1)$ any new processor failure discovered by applying the fault detection rules FD0–FD3 described in section 6.3 to $\texttt{tree}_i$ after the recording and masking steps have taken place. Since, as discussed later on, the fault detection rules will be computable in a fairly efficient manner, this whole step is feasible. We remark that it would be possible to use the new failures detected in the last step in order to mask additional nodes, and then perhaps perform the fault detection step again. Indeed, this process could be repeated until no new failures would be discovered. For the sake of simplicity, we choose not to do so. The failures discovered in round r will affect processors' messages and processing from round $r+1$ on.

Faulty processors will be assumed to be discovered according to a set of sound fault discovery rules. The only thing we require of this set of rules is that it should include the rules FD0–FD3 described in section 6.3. The soundness of the rules implies that one invariant of our algorithm will be the following.

- Soundness: If i and j are nonfaulty, then $j \notin \mathcal{F}_i(r)$ holds for all rounds r.

Given the soundness invariant, the sending and the recording and masking rules guarantee that $\texttt{tree}_i(\sigma) = \texttt{tree}_j(\sigma)$ will hold for every correct node σ of depth at most $t + 1 - \Delta$, and nonfaulty processors i and j. It follows that the values of the function $\texttt{right}^F$ will continue to be independent of the tree in which they are computed.

Prediction will be handled by a set of fixing rules Fx1–Fx3 described in section 6.2. These rules will determine when a node σ is said to be *fixed to value v in* $\texttt{tree}_i$. Recall that we defined a node σ to be *closed in* $\texttt{tree}_i$ at the end of round r if either σ or one of its ancestors is fixed in $\texttt{tree}_i$ at that point.

Finally, we shall use a simple rule for deciding on a value and for halting in the basic Δ-EIG protocol.

- Deciding: When the root λ becomes fixed to a value v in $\texttt{tree}_i$, processor i decides on value v.
- Halting: Processor i continues to record information, perform fault detection, and report on values until the end of round $t + 1 - \Delta$, at which point it halts.

We shall show in Lemma 6.7 and Corollary 6.9 that the root cannot be fixed both to 0 and to 1, and that the root is guaranteed to be fixed by the end of round $t+1-\Delta$. As a result, the decision rule given above is well defined and will guarantee that i will decide on a value.

To complete the description of Δ-EIG, we need to describe the rules by which nodes are fixed to values, and the manner in which processors perform fault discovery. This will be the subject of the next two subsections.

6.2. Fixing nodes to values. We now define when a node σ is *fixed* in $\texttt{tree}_i$ to a value v. This will be a major component in our protocol, and the properties of fixing, which we discuss below, will be instrumental in the development of the algorithm. We start with a fairly abstract definition of the fixing rules, relative to a function $F : [t] \to [n]$. We shall call such a function F *admissible* if it satisfies that

(i) $F(0) = F(1) = t + 1$, and

(ii) $t + r - 1 \geq F(r) \geq t - r + 2$ for $r \geq 2$.

In the sequel, we shall restrict our attention to admissible functions. A considerable amount of our analysis will be valid for admissible functions in general. Only in section 7 will we choose a particular admissible function to be used in our final protocol. We remark that admissible functions do not necessarily conform to the conditions of Lemma 5.3. As we shall see, the fact that we shall be fixing values of nodes before the full tree is developed will allow us to go beyond the bounds of Lemma 5.3. The role of Lemma 5.3 is in motivating the development of admissible functions and our fixing rules. It will not play a role in the correctness of our protocol in the end.

The bounds used in the definition of admissible functions were chosen so that they would match the following fixing rules. Formally, a node σ becomes *fixed in* $\texttt{tree}_i$ *to value* v at the end of round r if σ was not closed at the end of round $r-1$, and one of the following rules applies:

Fx1: $r = |\sigma| = t + 1 - \Delta$ and $\texttt{tree}_i(\sigma) = v$.

Fx2: $r = |\sigma| + 1$, $\texttt{par}(|\sigma|) = v$ and

$$\texttt{tree}_i(\sigma j) = v \text{ for at least } \begin{cases} n - t \text{ nodes } \sigma j & \text{if } \sigma = \lambda; \\ n - t - 1 \text{ nodes } \sigma j & \text{if } |\sigma| = 1; \text{ and} \\ n - t - 2 \text{ nodes } \sigma j & \text{if } |\sigma| \geq 2. \end{cases}$$

Fx3: Rules Fx1 and Fx2 do not apply, and either

(a) $\texttt{par}(|\sigma|) = v$ and at least $F(|\sigma|)$ of the σj's are fixed to v; or

(b) $\texttt{par}(|\sigma|) = 1 - v$ and at least $n - |\sigma| - F(|\sigma|) + 1$ (i.e., all but at most $F(|\sigma| - 1)$ of the σj's are fixed to v.

We remark that a naive bottom-up computation based on Fx1–Fx3 can be used to determine all of the fixed nodes and the values they are fixed to in a given EIG tree. Such a computation requires a number of steps at most linear in the size of the tree. The properties of the fixing rules Fx1–Fx3 will play a major role in the correctness of our ultimate protocol. We now consider some of these properties.

One immediate consequence of the definition of the above fixing rules is the following.

LEMMA 6.1. *Let F be admissible and let i be a nonfaulty processor. If all nonfaulty processors vote* 0*, then the root λ is fixed to* 0 *in* $\texttt{tree}_i$ *by the end of round* 1*. Similarly, if all nonfaulty processors vote* 1*, then the root λ is fixed to* 1 *in* $\texttt{tree}_i$ *by the end of round* 2*.*

Proof. Recall that $\texttt{par}(|\lambda|) = 0$. First assume that all nonfaulty processors vote 0. It follows that at least $n - t$ correct children of λ in $\texttt{tree}_i$ store 0 at the end of round 1, and as a result, by rule Fx2, the root is fixed to 0 at that point. Now assume that all nonfaulty processors vote 1. The root λ has at least $n - t$ correct children. Let σ denote a correct child of the root λ. Thus, at the end of round 1 we have that $\texttt{tree}_i(\sigma) = 1$. The root is not fixed yet. At the end of round 2, however, $\texttt{tree}_i(\sigma j) = 1$ for at least $n - t - 1$ children of σ. Since $\texttt{par}(|\sigma|) = 1$, it follows from the second clause of Fx2 that σ will be fixed to 1 at the end of round 2. We thus obtain that at least $n - t$ children of λ are fixed to 1 by the end of round 2. Recall that if F is admissible, then $F(0) = t + 1$. Hence, by rule Fx3 we now have that the root becomes fixed to 1 at the end of round 2, since that $F(|\lambda|) = F(0) = t + 1$, and $n - |\lambda| - F(|\lambda|) + 1 = n - (t + 1) + 1 = n - t$. □

Lemma 6.1 essentially takes care of the validity problem for Δ-agreement in the case in which all nonfaulty processors have initial votes of 0. The lemma states that in this case the root in all nonfaulty processors' trees will fix to 0 by the end of round 1, and by the decision clause of the Δ-EIG protocol, all nonfaulty processors

will decide 0 at that point. In the sequel, we shall therefore concern ourselves with the case in which at least one nonfaulty processor votes 1. Hence, we will be able to assume that the number of initially disabled processors is at least Δ.

We now wish to use the fixing rules to show that they guarantee that nodes of the righteous subtree end up being fixed to the desirable values. In Δ-agreement, however, we have processors that are initially disabled, that resemble nonfaulty processors in the fact that they cannot corrupt nodes or otherwise misbehave. As a result, finding a node corresponding to an initially disabled processor on a path from the root is quite analogous to finding a correct node. Following this observation, we shall therefore replace the definition of the righteous subtree with the analogous tree (which we shall call the *safe subtree*), and perform our analysis with respect to the new tree. Formally, we proceed as follows. Let $\mathcal{D}$ denote the set of initially disabled processors. We call a node σ *safe* if (i) σ is either righteous or corresponds to a processor in $\mathcal{D}$, and (ii) none of σ's ancestors are righteous, and none of them correspond to processors in $\mathcal{D}$. Since $\#\mathcal{D} \geq \Delta$ by definition of Δ-agreement, at most $t - \Delta$ processors labelling edges on a path from the root can be faulty but not from $\mathcal{D}$. It follows that every path of $t+2-\Delta$ nodes leading from the root must contain a safe node. In particular (since $|\lambda| = 0$), if σ is safe, then $|\sigma| \leq t + 1 - \Delta$. Since, by definition, there can be at most one safe node on every path from the root to leaves of the EIG tree, we obtain that the safe nodes form a cut in the tree. Let us denote this cut by C_s. We define the *safe subtree* of an EIG tree in an execution of Δ-EIG to consist of C_s and all ancestors of nodes in C_s. We remark that $|\sigma| = t + 1 - \Delta$ can hold for a node of the safe subtree only if σ is a safe node, and hence is either righteous or (initially) disabled. Finally, we shall use $h_s(\sigma)$, for a node σ of the safe subtree, to denote the length of the maximal path from the node σ to some node in C_s. More formally, for nodes σ in the safe subtree, we define

$$h_s(\sigma) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } \sigma \in C_s; \text{ and} \\ 1 + \max_j h_s(\sigma j) & \text{otherwise.} \end{cases}$$

One somewhat technical property that will serve us in the sequel is the following. (In this and all other statements in this paper, references to EIG trees such as $\texttt{tree}_i, \texttt{tree}_j$ and $\texttt{tree}_z$ are made only for nonfaulty processors; the assumption that i, j, or z is nonfaulty will be implicit.)

LEMMA 6.2. *Let F be an admissible function, and let σ be a node of the safe subtree such that $0 < |\sigma| < t+1-\Delta$, and all nonfaulty processors that do not appear in σ report the value v for σ. Then*

(a) *If $v = \texttt{par}(|\sigma|)$ then σ is closed in $\texttt{tree}_i$ by the end of round $|\sigma| + 1$ at the latest*; *while*

(b) *If $v \neq \texttt{par}(|\sigma|)$ then σ is closed in $\texttt{tree}_i$ by the end of round $\min(|\sigma|+2, t+1-\Delta)$ at the latest.*

Moreover, if σ is fixed in $\texttt{tree}_i$, then it is fixed to v.

Proof. We prove the claim separately for different depths of $|\sigma|$.

(i) For $t - \Delta > |\sigma| \geq 1$: We argue by cases depending on whether or not $v = \texttt{par}(|\sigma|)$.

(a) Assume $v = \texttt{par}(|\sigma|)$. If σ is closed by the end of round $|\sigma|$ then σ does not become fixed in $\texttt{tree}_i$ and we are done. Otherwise, σ has at least $n - t - 1$ correct children all of which store v by the end of round $|\sigma| + 1$. By rule Fx2 we thus obtain that σ is fixed to v at the end of round $|\sigma| + 1$.

(b) Now assume $v \neq \texttt{par}(|\sigma|)$. The node σ cannot be fixed to $v \neq \texttt{par}(|\sigma|)$ by rule Fx2, since this rule only allows fixing to $\texttt{par}(|\sigma|)$. If σ is closed by the end of

round $|\sigma|+1$, then σ does not become fixed in $\texttt{tree}_i$ and we are done. We shall show that if σ is not closed by the end of round $|\sigma|+1$ then σ becomes fixed to v by the end of round $|\sigma|+2$. Let σj be a correct child of σ. In particular, $\texttt{tree}_i(\sigma j) = v$, and all correct children of σj store v in $\texttt{tree}_i$ as well. Notice that $\texttt{par}(|\sigma j|) = v$ and $|\sigma j| \geq 2$. In addition, since σ is a node of the safe subtree and $|\sigma| < t - \Delta$, the node σj has at least $n - t - 2$ correct children, and they all store v. It follows that Fx1 does not apply to σj, and σj becomes fixed to v at the end of round $|\sigma|+2$ for all correct nodes σj. Recall that there are at least $n - t - 1$ such correct nodes σj. Let $r = |\sigma|$, and recall that we are assuming that $F(r) \geq t - r + 2$.[4] Thus, in particular, $n - r - F(r) + 1 \leq n - r - t + r - 2 + 1 = n - t - 1$. It now follows by Fx3(b) that σ is fixed to v in $\texttt{tree}_i$ by the end of round $|\sigma|+2$.

(ii) For $t - \Delta = |\sigma|$: If σ is closed at the end of round $|\sigma| = t - \Delta$, then σ does not become fixed in $\texttt{tree}_i$ and we are done. Otherwise, if $v = \texttt{par}(|\sigma|)$ then σ becomes fixed by rule Fx2 to v as in the case of $t - \Delta > |\sigma|$ described above. If $v \neq \texttt{par}(|\sigma|)$ then all of σ's children are fixed in $\texttt{tree}_i$ at the end of round $|\sigma|+1$. Moreover, at least $n - t - 1$ correct children σj of σ are fixed to v. As in the case of $|\sigma| > t - \Delta$ and $v \neq \texttt{par}(|\sigma|)$ we have that σ is fixed to v in $\texttt{tree}_i$ by rule Fx3(b). □

Lemma 6.2 immediately provides us with a number of useful corollaries. Essentially, Lemma 6.2 implies that nodes corresponding to initially disabled processors and righteous nodes are guaranteed to close quickly given our fixing rules.

COROLLARY 6.3. *Let F be admissible, and let $\sigma = \tau z$ be a node of the safe subtree such that $|\sigma| < t + 1 - \Delta$ and z is disabled by the end of round $|\sigma|$. Then σ is closed in $\texttt{tree}_i$ by the end of round $|\sigma|+1$ at the latest. Moreover, if σ becomes fixed to a value v in $\texttt{tree}_i$, then $v = \texttt{par}(|\sigma|) = \texttt{right}^F(|\sigma|)$.*

Proof. Since $|\sigma| < t + 1 - \Delta$, rule Fx1 does not apply to fixing σ. If σ is closed in $\texttt{tree}_i$ by the end of round $|\sigma|$, we are done. Otherwise, since we have that z is disabled by the end of round $|\sigma|$ and $\sigma = \tau z$, all nonfaulty processors j issue a $\texttt{mask}(j, z)$ report by round $|\sigma|+1$ at the latest. As a result, all nonfaulty processors are considered to be reporting $v = \texttt{par}(|\sigma|)$ for $\sigma = \tau z$. Lemma 6.2 now implies that σ is closed in $\texttt{tree}_i$ by the end of round $|\sigma|+1$, and, if it is fixed in $\texttt{tree}_i$, it is fixed to $v = \texttt{par}(|\sigma|)$. Since at least $n - t > n - t - 1 \geq F(|\sigma|)$ righteous children of τz in $\texttt{tree}_i$ store the value $\texttt{par}(|\sigma|)$, we obtain that $\texttt{par}(|\sigma|) = \texttt{right}^F(\sigma)$ and we are done. □

In particular, Corollary 6.3 implies that all nodes corresponding to initially disabled processors become closed within one round, and can fix only to their masked value. A similar situation holds with respect to righteous nodes as shown in the following corollary.

COROLLARY 6.4. *Let F be admissible, and let σ be a safe node (and hence a leaf of the safe subtree). Then σ is closed in $\texttt{tree}_i$ by the end of round $\min(|\sigma|+2, t+1-\Delta)$. Moreover, if σ becomes fixed to a value v in $\texttt{tree}_i$, then $v = \texttt{tree}_i(\sigma) = \texttt{right}^F(\sigma)$.*

Proof. First notice that if $|\sigma| = t + 1 - \Delta$ and σ is not closed by the end of round $t - \Delta$, then σ is fixed in $\texttt{tree}_i$ by rule Fx1 to $\texttt{tree}_i(\sigma) = \texttt{right}^F(\sigma)$ and we are done. If $|\sigma| = t + 1 - \Delta$ and σ is closed in $\texttt{tree}_i$ by the end of round $t - \Delta$, then σ is not fixed in $\texttt{tree}_i$ and we are done. Otherwise, we have that $|\sigma| < t+1-\Delta$. Let $\sigma = \tau z$, and assume that σ is not closed by the end of round $|\sigma|$. If σ is safe because $z \in \mathcal{D}$, then all processors report $\texttt{par}(|\sigma|)$ for σ in round $|\sigma|+1$ and we are done by Lemma 6.2. We may thus assume that σ is righteous, then every nonfaulty processor j reports $\texttt{tree}_j(\sigma)$ for σ. Since we are assuming that z is nonfaulty, then, by

[4] This is where we use the lower bound specified in the definition of admissible functions.

soundness of the fault detection module and the recording condition, every nonfaulty processor j would store the value that z reports for τ in $\texttt{tree}_j(\sigma) = \texttt{tree}_j(\tau z)$. Moreover, a nonfaulty processor z would report the same value for τ to all nonfaulty processors j. Thus, we have that $\texttt{tree}_i(\sigma) = \texttt{tree}_j(\sigma)$ for every nonfaulty processor j, and we obtain that all nonfaulty processors report the value $\texttt{tree}_i(\sigma) = \texttt{right}^F(\sigma)$ for σ. The claim now follows by Lemma 6.2. □

An immediate consequence of Corollaries 6.3 and 6.4 follows.

LEMMA 6.5. *A node σ that is not closed in $\texttt{tree}_i$ by the end of round $|\sigma|+1$ is in the safe subtree.*

Proof. Corollaries 6.3 and 6.4 imply that every node τ on the safe cut C_s is closed by the end of round $|\tau|+2$. Since these nodes form a cut in the EIG tree, every node σ that is not in the safe subtree has an ancestor τ in C_s; moreover, $|\sigma|+1 \geq |\tau|+2$. It follows that a node σ that is not in the safe subtree must be closed by the end of round $|\sigma|+1$ at the latest, and the claim follows. □

The fixing rule Fx3 has the property that once all children of a node are fixed, so is the node itself. As a result, we obtain the following lemma.

LEMMA 6.6. *For each nonfaulty i, every node σ of the safe subtree is closed in $\texttt{tree}_i$ by the end of round $t+1-\Delta$.*

Proof. We prove the claim by induction on $h_s(\sigma)$ for nodes σ of the safe subtree. Recall that we have defined $h_s(\sigma)$ to be the height of σ in the safe subtree. In particular, we clearly have that $h_s(\sigma) \leq t+1-\Delta$ for every node σ of the safe subtree. If $h_s(\sigma) = 0$, then σ is a safe node, by definition of h_s. The claim now follows from Corollary 6.4. Now assume $h_s(\sigma) = k > 0$, and assume the claim holds for all nodes τ satisfying $h_s(\tau) < k$. In particular, every child σj of σ is in the safe subtree and satisfies $h_s(\sigma j) < h_s(\sigma) = k$. Thus, the inductive assumption implies that all of σ's children are closed by the end of round $t+1-\Delta$. If this is because σ or one of its ancestors are fixed, we are done. Otherwise, all of σ's children are fixed by the end of round $t+1-\Delta$. Recall that the number of σ's children is $n-|\sigma|$. If rule Fx3(a) does not apply, then fewer than $t+2-|\sigma|$ of the children of σ are fixed to the value $\texttt{par}(|\sigma|)$. It then follows that at least $n-|\sigma|-(t+1-|\sigma|) = n-t-1$ of the children of σ are fixed to $1-\texttt{par}(|\sigma|)$, so that σ is fixed to $1-\texttt{par}(|\sigma|)$ in $\texttt{tree}_i$ by rule Fx3(b). In either case we obtain that σ must also be fixed in $\texttt{tree}_i$, and we are done. □

A crucial property of our fixing rules is that a node σ can be fixed to at most one value in $\texttt{tree}_i$, as we now prove in the following lemma.

LEMMA 6.7. *If a node σ of the safe subtree is fixed to value v in $\texttt{tree}_i$, then σ is not fixed to $1-v$ in $\texttt{tree}_i$.*

Proof. We prove the claim by induction on $h_s(\sigma)$. The case $h_s(\sigma) = 0$ follows from Corollary 6.4. Assume that $h_s(\sigma) \geq 1$ and that the claim holds for all nodes τ of the safe subtree with $h_s(\tau) < h_s(\sigma)$. Hence, in particular, rule Fx1 does not apply to σ, and the claim holds for σ's children in $\texttt{tree}_i$. Assume σ is fixed by Fx2. The only value to which σ can be fixed by Fx2 is $\texttt{par}(|\sigma|)$. Moreover, by the definition of these rules, if Fx2 applies to σ in $\texttt{tree}_i$ then neither Fx1 nor Fx3 do. It follows that if σ is fixed by rule Fx2, then it is fixed to a unique value. Finally, assume σ becomes fixed in $\texttt{tree}_i$ by rule Fx3. In particular, Fx1 and Fx2 do not apply in the case of σ. The node σ has exactly $n-|\sigma|$ children σj. By definition, $h_s(\sigma j) < h_s(\sigma)$. Thus, by the inductive hypothesis, each of these children is fixed to at most one value. The total number of fixed children of σ that would be necessary for both Fx3(a) and Fx3(b) to apply is $t+2-|\sigma|+n-t-1 = n-|\sigma|+1$, which is more than the number of children of σ. It follows that only one of these rules can apply, so that σ can be fixed to at

most one value in $\texttt{tree}_i$. □

We are now ready to prove that nodes of the safe subtree can become fixed only to their $\texttt{right}^F$ values.

THEOREM 6.8. *If F is admissible and a node σ of the safe subtree is fixed to value v in $\texttt{tree}_i$, then $v = \texttt{right}^F(\sigma)$.*

Proof. We prove the claim by induction on $h_s(\sigma)$. The case of $h_s(\sigma) = 0$ follows directly from Corollary 6.4.

Assume that $h_s(\sigma) > 0$ and the claim holds for all children σj of σ. (By definition of h_s we have that $h_s(\sigma) > h_s(\sigma j) \geq 0$.) Notice that the rule Fx1 cannot apply to σ since Fx1 deals with nodes τ satisfying $|\tau| = t + 1 - \Delta$, and such a node is in the safe subtree only if it is a safe node, in which case $h_s(\tau) = 0$. Thus, $h_s(\sigma) > 0$, and only rules Fx2 and Fx3 can apply for fixing σ. Since σ is an internal node of the safe subtree, σ and all of its ancestors are incorrect nodes. As a result, at least $n-t$ of σ's children are correct, while at most $t - |\sigma|$ of them are faulty. Moreover, every correct child of σ is righteous. We now consider the ways in which σ can become fixed due to Fx2 and Fx3.

(i) Assume that $|\sigma| \geq 2$ and σ becomes fixed in $\texttt{tree}_i$ to value $v = \texttt{par}(\sigma)$ by rule Fx2. The definition of Fx2 implies that $\texttt{tree}_i(\sigma j) = v$ for at least $n - t - 2$ nodes σj. Since at most $t - |\sigma|$ of these may be incorrect, we obtain that at least $n-t-2-t+|\sigma| = n-2t+|\sigma|-2 \geq t+|\sigma|-1$ of these nodes are correct. For each such correct child σj we have that $\texttt{right}^F(\sigma j) = v$. Since F is admissible, we have that $F(|\sigma|) \leq t+|\sigma|-1$, and it follows from the definition of $\texttt{right}^F$ that $\texttt{right}^F(\sigma) = v$.[5] A similar argument applies for the cases of $|\sigma| \leq 1$. In these cases, $F(|\sigma|) = t + 1$, at least $n - t - |\sigma|$ of the children are fixed to v, and at most $t - |\sigma|$ of the children are incorrect. In both cases $n - t - |\sigma| - t + |\sigma| = n - 2t \geq t + 1 = F(|\sigma|)$ and we are done.

(ii) If σ is fixed in $\texttt{tree}_i$ to v by rule Fx3(a), then $\texttt{par}(|\sigma|) = v$ and we have by the inductive hypothesis for the $F(|\sigma|) \geq t + 2 - |\sigma|$ nodes of the form σj that are fixed to v in $\texttt{tree}_i$ that $\texttt{right}^F(\sigma j) = v$. Thus, by the definition of $\texttt{right}^F$ we have that $\texttt{right}^F(\sigma) = v$ as well. An analogous argument works for fixing based on Fx3(b). □

COROLLARY 6.9. *The root λ of $\texttt{tree}_i$ is fixed to value $\texttt{right}^F(\lambda)$ by the end of round $t + 1 - \Delta$.*

Proof. Lemma 6.6 implies that λ is closed in $\texttt{tree}_i$ by the end of round $t+1-\Delta$. Since λ has no ancestors, it must be fixed at that time. Theorem 6.8 implies that λ is fixed to $\texttt{right}^F(\lambda)$. □

THEOREM 6.10. *For any sound fault-detection module and admissible function F, the Δ-EIG protocol satisfies the decision, agreement, and validity properties.*

Proof. Decision follows immediately from Corollary 6.9 and Lemma 6.7. Recall from our discussion of $\texttt{right}^F$ that if σ is in the righteous subtree, then the value of $\texttt{right}^F(\sigma)$ is independent of the tree in which it is computed. Corollary 6.9 implies that every nonfaulty processor decides on $\texttt{right}^F(\lambda)$, and since λ is in the righteous subtree, we obtain agreement. We now argue why validity holds. All correct children of λ are righteous. If all nonfaulty processors j have the same initial value $v_j = v$, then $\texttt{right}^F(\sigma) = v$ for all $n - t$ righteous children σ of λ. By definition of $\texttt{right}^F$, this implies that $\texttt{right}^F(\lambda) = v$, and since $\texttt{right}^F(\lambda)$ is the value decided on, we obtain validity. □

[5] This is where we use the upper bound specified in the definition of admissible functions.

6.3. Fault detection in Δ-EIG. We now turn to describing the fault discovery rules we shall use in the instances of Δ-agreement for the purposes of our final protocol. Notice that all of the results regarding fixing that we have seen above depend only on the masking and soundness rules, which state that initially detected failures must be masked to the favored value, and nonfaulty processors are not masked by other nonfaulty processors. Thus, we have a considerable amount of freedom in introducing fault discovery rules without affecting the correctness of the protocol.

We find it convenient to consider the notion of a node σ being *committed to value* v *in* $\texttt{tree}_i$. Intuitively, $\sigma = \tau z$ will be committed to v in $\texttt{tree}_i$ only if i has a proof that at least one nonfaulty processor either has received a report of v for τ from z, or is masking z. Formally, we say that a node $\sigma \neq \lambda$ is committed to v in $\texttt{tree}_i$ if one of the following holds:

C1: $\texttt{tree}_i(\sigma) = v$;

C2: $\texttt{tree}_i(\sigma j) = v$ for at least $\min(t+1, t+3-|\sigma|)$ nodes σj;

C3: σ is not closed in $\texttt{tree}_i$ at the end of round $|\sigma|+1$ and σ is not fixed to $1-v$ at the end of round $|\sigma|+2$.

As in the case of fixing, a naive linear-time computation based on C1, C2, and C3 is easily seen to suffice for determining all of the commitments of nodes to values in a given EIG tree. We remark that these rules will only be applied to nodes of the safe subtree. For such nodes, the bound of C2 guarantees that one of the children σj with $\texttt{tree}_i(\sigma j) = v$ is correct.

The main use we have for the notion of commitment is captured in the following lemma.

LEMMA 6.11. *Let τ be a node of the safe subtree. If a child τz of τ is committed to v in $\texttt{tree}_i$ by the end of round r, then, for at least one nonfaulty processor j, either $z \in \mathcal{F}_j(|\tau z|)$ or j received a report of v from z for τ.*

Proof. If τz is committed to v in $\texttt{tree}_i$ by C1, then the claim holds trivially for $j = i$, since $\texttt{tree}_i(\tau z) = v$ only if either i received a report of v for τ from z, or $v = \texttt{par}(|\tau|)$ and i is masking z in round $|\tau z|$. Assume that τz is committed to v in $\texttt{tree}_i$ by C2. Since τ is a node of the safe subtree, all, except possibly for the last, members of the sequence τ are faulty processors. It follows that the number of incorrect children of τz is at most $t - |\tau| + 1$ $(= t + 2 - |\tau z|)$. We thus obtain that if C2 applies, then at least one of the children $\tau z j$ of τz with $\texttt{tree}_i(\tau z j) = v$ must be a correct node. But $\texttt{tree}_i(\tau z j) = v$ for a correct node $\tau z j$ only if either $z \in \mathcal{F}_j(|\tau z|)$ (in which case j sends a $\texttt{mask}(j, z)$ message no later than in round $|\tau z| + 1$), or if j received a report of v for τ from z. We are left with the case of commitment to v due to C3. First notice that if τz is righteous and committed to v in $\texttt{tree}_i$ by C3, then it is already committed to v in $\texttt{tree}_i$ by C2. This is because if τz is not closed in $\texttt{tree}_i$ at the end of round $|\tau z| + 1$, then it is committed in $\texttt{tree}_i$ to value $w = \texttt{tree}_i(\tau z)$, and by Corollary 6.4, it will not not be committed to $1 - w$ by C3. Thus, we may assume that τz is not righteous. In particular, it has at least $n - t$ righteous children. It suffices to show that at least one nonfaulty processor j reports v for τz. Assume not. Then all nonfaulty processors report $1 - v$ for τz. It now follows by Lemma 6.2 that τz must be fixed to $1 - v$ by the end of round $|\tau z| + 2$ if it was not closed by the end of round $|\tau z| + 1$. This contradicts the assumption that τz is committed to v in $\texttt{tree}_i$ by C3. □

LEMMA 6.12. *Assume that $t \geq 3$. If a node $\sigma \neq \lambda$ of the safe subtree is ever fixed to value v in $\texttt{tree}_i$, then σ is committed to v in $\texttt{tree}_i$ by the end of round $|\sigma| + 2$.*

Proof. Assume σ is fixed to v in $\texttt{tree}_i$ by the end of round $|\sigma| + 2$. If σ is fixed

by rule Fx1, then by C1 it is also committed to v in $\texttt{tree}_i$. If it is fixed by Fx2, then it is committed to v by C2, since Fx2 implies that $\texttt{tree}_i(\sigma j) = v$ for at least $n-t-2$ nodes σj. Given that $t \geq 3$ and $|\sigma| \geq 1$, we have

$$n - t - 2 \geq 2t + 1 - 2 = 2t - 1 \geq t + 3 - 1 = t + 2 \geq t + 3 - |\sigma|.$$

Finally, assume that σ is fixed to v by Fx3. In particular, σ is not closed in $\texttt{tree}_i$ by the end of round $|\sigma| + 1$, and it becomes fixed to v no earlier than the end of round $|\sigma| + 2$. Lemma 6.7 implies that σ can be fixed to at most one value in $\texttt{tree}_i$, so that σ is not fixed to $1 - v$ at the end of round $|\sigma| + 2$, and by C3 it is committed to v in $\texttt{tree}_i$ at that point. □

Notice that for every node $\sigma \neq \lambda$ of the safe subtree, there must be at least one value $v \in \{0, 1\}$ such that at least $t + 1$ (and hence $\geq t + 2 - |\sigma|$) nonfaulty processors report v for σ. We say that σ is then *publicly committed* to such a value v. If σ is publicly committed to v, then σ becomes committed by C2 to v in $\texttt{tree}_i$ for all trees $\texttt{tree}_i$ in which values of children of σ are stored. A variant of Lemma 6.12 that applies to public commitment and will be useful in the sequel is the following lemma.

LEMMA 6.13. *Let σ be a node of the safe subtree of depth $2 \leq |\sigma| \leq t - \Delta$, let $v = \texttt{par}(|\sigma|)$, and let σ be fixed in $\texttt{tree}_i$ to $1 - v$ (the "disfavored" value). Then σ is publicly committed to $1 - v$.*

Proof. By Theorem 6.8, a node σ of the safe subtree can be fixed only to $\texttt{right}^F(\sigma)$. Given our definition of publicly committed, we have that a node must be publicly committed to at least one value among $0, 1$. However, if σ were publicly committed to $v = \texttt{par}(|\sigma|)$, then by definition of $\texttt{right}^F$ we would have that $\texttt{right}^F(\sigma) = \texttt{par}(|\sigma|) \neq 1 - v$. It follows that σ must be publicly committed to $1 - v$. □

We are now ready to present our fault detection rules. Intuitively, a nonfaulty processor i discovers that a processor is faulty when there is "enough" evidence that the processor has sent conflicting values to other correct processors. This evidence may be gathered when the messages of the children of a node corresponding to the faulty processor are received. Formally, processor i will detect z at the end of round as being faulty if one of the following holds:

FD0: z sends i an ill-formatted message in round r.

FD1: By the end of round r, processor i has received $\texttt{mask}(j, z)$ reports from at least $t + 1$ distinct processors j.

FD2: By the end of round r, some node τz that was not closed in $\texttt{tree}_i$ by the end of round $|\tau z|$ is committed both to 0 and to 1 in $\texttt{tree}_i$.

FD3: By the end of round r, for some node $\tau a z$ and value v such that (a) $r = |\tau a z| + 1$, and (b) $\tau a z$ is not closed in $\texttt{tree}_i$ by the end of round r; we have that

(i) $\tau a z$ is committed to v in $\texttt{tree}_i$;

(ii) at least $2(t + 1 - |\tau a|) + 1$ of the nodes $\tau a j$ are committed to $1 - v$ in $\texttt{tree}_i$ by the end of round r; and

(iii) z does not mask a in round $|\tau a z| + 1$. (Namely, z did not send a $\texttt{mask}(z, a)$ report to i in the first r rounds.)

The motivation for the first three rules FD0–FD2 is fairly intuitive and straightforward. Similar rules have appeared in earlier work in the literature. The fourth rule, FD3, is of a new type. It is tailor made for handling a specific type of corruption, called cross corruption, that we will consider in detail in section 7. Intuitively, FD3 can be thought of as detecting a *crime of omission*. It applies when the detecting processor i has a proof that, had z been nonfaulty, then z would have detected another processor a as being faulty due to FD2 in round $r - 1$. As a result, z should have

issued a $\texttt{mask}(z, a)$ report no later than in round r. By rule FD3, i discovers z as being faulty once z fails to issue a $\texttt{mask}(z, a)$ in time. Figure 2 illustrates this scenario.

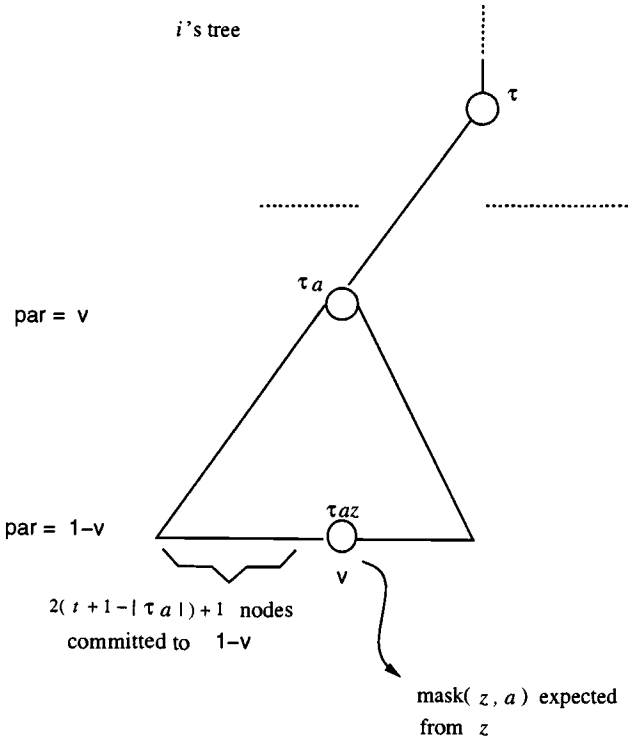


FIG. 2. *Discovering a fault using* FD3.

LEMMA 6.14. *The rules* FD0–FD3 *are sound. That is, if the fault detection module is sound for all nonfaulty processors through the end of round* $r - 1$*, then every processor that is added to* $\mathcal{F}_i(r)$ *according to one of these rules is faulty.*

Proof. The soundness of FD0 is trivial, since a processor that sends an ill-formatted message is deviating from the protocol, and hence is faulty. If rule FD1 applies, then at least one of the $t + 1$ processors that report to i that they are masking z must be nonfaulty. The soundness of the rule now follows from our assumption about the soundness of the fault detection module in the first $r - 1$ rounds. We now turn to FD2. Notice that this rule cannot apply with respect to a node τz before the end of round $|\tau z| + 1$, since only one commitment value can be obtained in $\texttt{tree}_i$ before values for the children of τz are stored. By Lemma 6.5, if τz is not closed at the end of round $|\tau z|$, then τ is a node of the safe subtree. Since τz is committed both to 0 and to 1, Lemma 6.11 implies that either at least one nonfaulty processor is masking z in round $|\tau z|$, or z sent conflicting reports of both 0 and 1 for τ to two different nonfaulty processors. Having assumed that the fault detection module was sound in the first $r - 1$ rounds we obtain that, in either case, z must be faulty, and hence FD2 is sound.

Finally, let us consider FD3. Roughly speaking, the soundness of FD3 is based on FD2 and the fact that if (i) and (ii) hold, then i can determine that z must have had enough information for detecting a as faulty using FD2 in round $|\tau a z|$. If z did not

react accordingly, then it must be faulty. We now formalize this intuition. Assume that z is nonfaulty, and conditions (i) and (ii) apply. Moreover, assume that z does not issue a $\texttt{mask}(z, a)$ report by the end of round $|\tau az|$. Since τaz is not closed by the end of round $|\tau az| + 1$, then by Lemma 6.5 the node τaz is in the safe subtree, and hence all of the processors in the sequence τa are faulty. It follows that at most $t - |\tau a|$ of the nodes τaj that are committed to $1 - v$ in $\texttt{tree}_i$ by the end of round $|\tau az| + 1$ can be incorrect. $t - |\tau a|$ can be incorrect. Since their total number is, by assumption, at least $2(t + 1 - |\tau a|) + 1$, it follows that at least

$$2(t + 1 - |\tau a|) + 1 - (t - |\tau a|) = t + 3 - |\tau a|$$

of these nodes are correct, and, in particular, must appear in $\texttt{tree}_z$. It thus follows by rule C2 that τa must be committed to $1 - v$ in $\texttt{tree}_z$ by the end of round $|\tau az|$. However, since τaz is committed to v in $\texttt{tree}_i$, (and z is nonfaulty) it must be the case that τa is committed to v in $\texttt{tree}_z$ as well by the end of round $|\tau az|$. Thus, by rule FD2 we obtain that z must detect a as faulty in round $|\tau az|$. Moreover, by the masking behavior rule, in round $|\tau az|+1$, processor z must report that it is masking a, if it has not done so in an earlier round. If z fails to do so, then z must be faulty as determined by FD3. □

Given these fault discovery rules, we can now turn to study the conditions under which nodes can be corrupted in instances of Δ-EIG. In addition, we shall be interested in the relationship between the corrupted nodes and the size of the EIG tree constructed.

6.4. Corrupting nodes in Δ-EIG. Formally, in an execution of the Δ-EIG protocol we define a node σ to be *corrupted in* $\texttt{tree}_i$ if σ is not closed in $\texttt{tree}_i$ by the end of round $|\sigma| + 2$.

A node σ is said to be *universally corrupted* if it is corrupted in $\texttt{tree}_i$ for all nonfaulty processors i. The following three lemmas show that, in order to cause lasting trouble, a node must be universally corrupted.

LEMMA 6.15. *Let i and j be nonfaulty processors, and let σ be a node of the safe subtree. If σ is fixed in $\texttt{tree}_i$ by rule Fx2, then σ is closed in $\texttt{tree}_j$ by the end of round $|\sigma| + 3$.*

Proof. The claim follows immediately for righteous nodes σ by Corollary 6.4. We shall henceforth consider the case in which σ is not righteous, so that at most $t - |\sigma|$ of its children are faulty. We prove the claim for $|\sigma| \geq 2$; the modifications for the cases $|\sigma| = 0$ and $|\sigma| = 1$ are simple and left for the reader. Assume that σ is fixed in $\texttt{tree}_i$ by rule Fx2, and that σ is not closed in $\texttt{tree}_j$ by the end of round $|\sigma| + 2$. Since σ is fixed to v in $\texttt{tree}_i$ by Fx2 in round $|\sigma| + 1$, we have that at least $n - t - 2 - (t - |\sigma|) = n - 2t + |\sigma| - 2 \geq t + |\sigma| - 1$ of σ's children in $\texttt{tree}_j$ are righteous children that store v. If σ is not closed in $\texttt{tree}_j$ beforehand, then Corollary 6.4 implies that these nodes will all be fixed to v in $\texttt{tree}_j$ by the end of round $|\sigma| + 3$. Since $F(|\sigma|) \leq t + |\sigma| - 1$ we have that σ becomes fixed to v in $\texttt{tree}_j$ by Fx3(a) at the end of round $|\sigma| + 3$. □

As a consequence of Lemma 6.15, a straightforward induction yields the following.

COROLLARY 6.16. *Let i and j be nonfaulty processors, and let σ be a node of the safe subtree. If σ is closed in $\texttt{tree}_i$ by the end of round r, then it is closed in $\texttt{tree}_j$ by the end of round $r + 2$.*

COROLLARY 6.17. *Let i be a nonfaulty processor, and let τ be a node that is not universally corrupted. Then τ is closed in $\texttt{tree}_i$ by the end of round $|\tau| + 4$.*

Proof. By definition of corruption, a node τ that is not universally corrupted must be closed in $\texttt{tree}_j$ for some nonfaulty j by the end of round $|\tau|+2$. By Corollary 6.16 we have that τ is closed in $\texttt{tree}_i$ by the end of round $|\tau|+4$. □

As in the case of $n > 4t$, there is a close relationship between corrupted nodes and failure detection. Indeed, an immediate consequence of the fault discovery rule FD2 and the commitment rule C3 is the fact that a processor that corrupts a node in $\texttt{tree}_i$ is discovered by i as being faulty. Formally, we have the following lemma.

LEMMA 6.18. *If a node τz is corrupted in $\texttt{tree}_i$, then $z \in \mathcal{F}_i(|\tau z|+2)$.*

Given Lemmas 6.2 and 6.18, we obtain the following relationship between universal corruption and disabled processors.

COROLLARY 6.19. *If a node τz is universally corrupted, then processor z is disabled from the end of round $|\tau z|+2$ on.*

Since corruption is determined within at most two rounds, Corollary 6.19 implies that a faulty processor can universally corrupt nodes in at most two different rounds. Thus, our situation resembles that of the $n > 4t$ case with majority, except that now the faulty processors are able to corrupt nodes in two consecutive rounds in some cases. This will force us to consider the possibility of cross corruption in section 7.

6.4.1. Waste. One consequence of Corollary 6.17 is that if no node σ of depth r is universally corrupted, then all such nodes are closed by the end of round $r+4$. Moreover, it is easy to see that if all nodes of depth r are closed, then so are all of their ancestors including the root. We thus have the following lemma.

LEMMA 6.20. *If no node σ of depth $|\sigma| = r$ is universally corrupted, then the root λ is closed in all processors' trees by the end of round $r+4$.*

We say that a processor z *universally corrupts a node τz at depth r* if the node τz is universally corrupted, and $|\tau z| = r$. Notice that whether z universally corrupts τz might depend on events that take place after round $r = |\tau z|$, such as what values other faulty processors report for τz, and who is masking z in round $r+1$. Intuitively, we can figure out whether z universally corrupted τz only in round $r+2$. Corollary 6.19 states that at most two rounds after a processor universally corrupts a node, this processor is disabled. Corollary 6.3 implies that a disabled processor cannot universally corrupt nodes. It follows that a processor can universally corrupt nodes in at most two (consecutive) rounds. However, it is not hard to see that a processor z cannot be the only processor universally corrupting nodes in a pair of consecutive rounds r and $r+1$.

LEMMA 6.21. *Assume that z universally corrupts nodes at depth r. If there are universally corrupted nodes at depth $r+1$, then there must be at least one processor $z' \neq z$ corrupting nodes either at depth r or at depth $r+1$.*

Proof. Assume that τxy is a node of depth $|\tau xy| = r+1$ that is universally corrupted. If no processor other than z universally corrupts nodes at depth $r+1$, we must have that $y = z$. By definition of the EIG tree, a node is a sequence of processor names without repetitions, and hence $x \neq z$. Since τxy is universally corrupted, it is not closed in any processor's tree by the end of round $|\tau|+4$. It follows that τx is also not closed in any processor's tree at that point. We conclude that τx was not closed in any tree at the end of round $|\tau|+3 = |\tau x|+2$, and hence was universally corrupted. Thus, $x \neq z$ universally corrupted nodes at depth r, and the claim follows. □

Lemma 6.20 implies that in order to keep the trees from closing, at least one processor must universally corrupt nodes at every depth. Moreover, Lemma 6.21 implies that at least two different processors must universally corrupt nodes in consecutive rounds. Finally, by Corollary 6.19, we know that two rounds after a processor universally corrupts nodes, this processor is disabled. It follows that, roughly speaking,

in order to keep the root from closing, at least one processor per round must become disabled. We now make a few definitions that will allow us to make this intuition into a precise statement, and will later help us in the analysis of monitor voting.

Let us denote by $\mathcal{D}(r)$ the set of processors that are disabled at the end of round r. We define the *deficit* at r, denoted $\texttt{deficit}(r)$, to be the number of processors that universally corrupt nodes at depth $r-1$, but are not disabled by the end of round r. Recall that every processor that universally corrupts a node at depth $r-1$ is detected by all nonfaulty processors as faulty (and hence disabled) by the end of round $r+1$. We thus have $\#\mathcal{D}(r+1) \geq \#\mathcal{D}(r) + \texttt{deficit}(r)$. We are almost ready to define the *waste* at r. Roughly speaking, the term *waste* comes from the idea that the nonfaulty processors are playing against an adversary.[6] This adversary is trying to enlarge the size of processors' trees without spending more than one disabled processor per round. The waste measures the extent by which the adversary has exceeded this allowance. Intuitively, the waste should be a measure, stated in terms of the number of disabled processors and the deficit, that will have the following properties:

1. As long as at least one node is universally corrupted in every round, the waste should be nondecreasing;

2. an appropriate form of monitor voting will guarantee that if the waste exceeds a certain constant threshold, then the agreement process will be halted; and

3. as long as the waste does not exceed the threshold of (2), then the number of universally corrupted nodes in the tree is polynomial.

To obtain this, we define the *waste* at the end of round r, denoted by $\texttt{Waste}(r)$, as follows:

$$\texttt{Waste}(r) \;=\; \#\mathcal{D}(r) - r + \texttt{deficit}(r) - \texttt{correction}(r).$$

The last term, $\texttt{correction}(r)$, is technically needed in order to compensate for cases in which the processors that form the deficit are able to corrupt nodes at depth r in addition to round $r-1$. In this case, we want to account one of these processors to the following round. Formally, we define $\texttt{correction}(r)$ as follows:

$$\texttt{correction}(r) = \begin{cases} 0 & \text{if } \texttt{deficit}(r) = 0; \\ 0 & \text{if } \texttt{deficit}(r) = 1 \text{ and only one processor universally} \\ & \qquad\qquad \text{corrupts nodes at depth } r-1; \quad \text{and} \\ 1 & \text{otherwise.} \end{cases}$$

In other words, the correction is 1 if either $\texttt{deficit}(r) \geq 2$, or if $\texttt{deficit}(r) = 1$, provided there is at least one processor universally corrupting nodes at depth $r-1$ in addition to the processor forming the deficit at r. In either case, we deduct a cost of 1 for the fact that some processor in the deficit had the option of causing harm at depth r as well as at depth $r-1$. The following lemma makes this claim precise.

LEMMA 6.22. *If* $\texttt{correction}(r) = 0$ *then no processor universally corrupts nodes both at depth* $r-1$ *and at depth* r.

Proof. We consider the two cases in which $\texttt{correction}(r) = 0$. For the first case, if $\texttt{deficit}(r) = 0$, then all processors that universally corrupt nodes in round $r-1$ are disabled by the end of round r, and hence cannot corrupt nodes at depth r. For the other case, assume that $\texttt{deficit}(r) = 1$ and let z be the processor forming the deficit at r. By definition, if $\texttt{correction}(r) = 0$ then z is the only processor that

[6] The notion of *waste* used here is close in spirit to, though technically quite different from, a similar notion introduced in the work of Dwork and Moses [12].

universally corrupts nodes at depth $r-1$. By Lemma 6.21, z cannot also universally corrupt nodes at depth r. □

We can now prove the following theorem.

THEOREM 6.23. *Let $r \geq 2$. If $\texttt{Waste}(r+1) < \texttt{Waste}(r)$ then no node of depth r is universally corrupted.*

Proof. We prove the contrapositive: assume that there is at least one universally corrupted node at depth r, and we shall show that $\texttt{Waste}(r+1) \geq \texttt{Waste}(r)$. Recall from the discussion above that $\#\mathcal{D}(r+1) \geq \#\mathcal{D}(r) + \texttt{deficit}(r)$. In addition, notice that, by definition of `correction`, we are guaranteed that $\texttt{deficit}(r') - \texttt{correction}(r') \geq 0$ for all r', and in particular we will have that $\texttt{deficit}(r+1) - \texttt{correction}(r+1) \geq 0$. We consider three cases:

(i) At least one processor universally corrupting nodes at depth $r-1$ also universally corrupts nodes at depth r. In this case, we have by Lemma 6.22 that $\texttt{correction}(r) = 1$. We thus have:

$$\begin{aligned}\texttt{Waste}(r+1) &= \#\mathcal{D}(r+1) - (r+1) + \texttt{deficit}(r+1) - \texttt{correction}(r+1) \geq \\ &\#\mathcal{D}(r+1) - (r+1) \geq \#\mathcal{D}(r) + \texttt{deficit}(r) - r - 1 = \\ &\#\mathcal{D}(r) - r + \texttt{deficit}(r) - \texttt{correction}(r) = \texttt{Waste}(r).\end{aligned}$$

(ii) The assumption of case (i) does not hold, and at least one processor z that universally corrupts nodes at depth r is disabled by the end of round $r+1$. Clearly, $z \notin \mathcal{D}(r)$, and by assumption we have that z is not one of the processors forming a deficit at r. Since z is disabled by the end of round $r+1$, we have that $\#\mathcal{D}(r+1) \geq \#\mathcal{D}(r) + \texttt{deficit}(r) + 1$, and hence we are guaranteed that $\texttt{Waste}(r+1) \geq \texttt{Waste}(r)$ by:

$$\begin{aligned}&\texttt{Waste}(r+1) \geq \#\mathcal{D}(r+1) - (r+1) \geq \#\mathcal{D}(r) + \texttt{deficit}(r) + 1 - (r+1) = \\ &\#\mathcal{D}(r) + \texttt{deficit}(r) - r \geq \#\mathcal{D}(r) - r + \texttt{deficit}(r) - \texttt{correction}(r) = \texttt{Waste}(r).\end{aligned}$$

(iii) The assumptions of cases (i) and (ii) do not hold. Thus, no processor universally corrupts nodes both at depth $r-1$ and at depth r, and no processor that universally corrupts nodes at depth r is disabled by the end of round $r+1$. It follows that only processors forming the deficit at $r+1$ universally corrupt nodes at depth r. In this case, if $\texttt{deficit}(r+1) \geq 2$, then $\texttt{deficit}(r+1) - \texttt{correction}(r+1) \geq 1$, and we have that that $\texttt{Waste}(r+1) \geq \#\mathcal{D}(r) + \texttt{deficit}(r) + 1 - (r+1)$ and hence $\texttt{Waste}(r+1) \geq \texttt{Waste}(r)$ as in the previous case. Finally, if $\texttt{deficit}(r+1) = 1$ then the assumptions imply that the processor forming the deficit at $r+1$ is the only processor universally corrupting nodes at depth r. By the definition of `correction` we thus have that $\texttt{correction}(r+1) = 0$, and we again obtain that $\texttt{deficit}(r+1) - \texttt{correction}(r+1) \geq 1$, so that $\texttt{Waste}(r+1) \geq \texttt{Waste}(r)$ and we are done. □

Theorem 6.23 will be instrumental in the correctness of our ultimate algorithm. We shall design the monitor voting scheme in such a way that once the waste becomes large enough (which in our case will mean at least two), an appropriate monitor decision process will "fire," thereby causing the processors all to decide on a default value and halt.

In section 7.3 we shall demonstrate how it is possible to stop interacting about a subtree of the EIG tree after it becomes closed. As a result, a fundamental parameter determining the size of the processors' trees will be the number of universally corrupted nodes in the tree. Theorem 6.23 allows us to formalize the idea that there is a close relationship between the waste of an execution and the number of universally

corrupted nodes in the tree. Recall, for example, that if there is only one processor universally corrupting nodes at any given depth, then the total number of universally corrupted nodes is no greater than t. In order for this number to grow, it is necessary for there to be levels of the tree at which two or more processors universally corrupt nodes. Notice, however, that if three or more processors universally corrupt nodes at depth r, then Corollary 6.19 implies they all will be disabled by the end of round $r+2$, and as a result we would have that $\texttt{Waste}(r+2) > \texttt{Waste}(r)$. As we are going to start with a waste at $r = 2$ of at least -1, it will follow that after a constant number of such rounds, the monitors will detect a problem and stop the growth of the EIG tree.

The only situation in which the number of universally corrupted nodes can grow more than in a linear fashion, and the waste need not increase, is in the case of *cross corruption*. This is a situation in which two processors, say a and b, universally corrupt nodes at depth r, and they continue to be the only processors to universally corrupt nodes at depth $r+1$. Specifically, if a corrupted a node τa at depth r and b corrupted τb, then at depth $r+1$ we will find b universally corrupting τab, while a corrupts τba. See Figure 3 for an illustration of this situation. In this fashion, it is possible to double the number of universally corrupted nodes of depth $r+1$ compared to the number of corrupted nodes of depth $r-1$, without increasing the waste.[7]

The next section is devoted to cross corruption. In particular, we shall devise an admissible resolve function that will restrict the number of times at which cross corruption need not increase the waste.

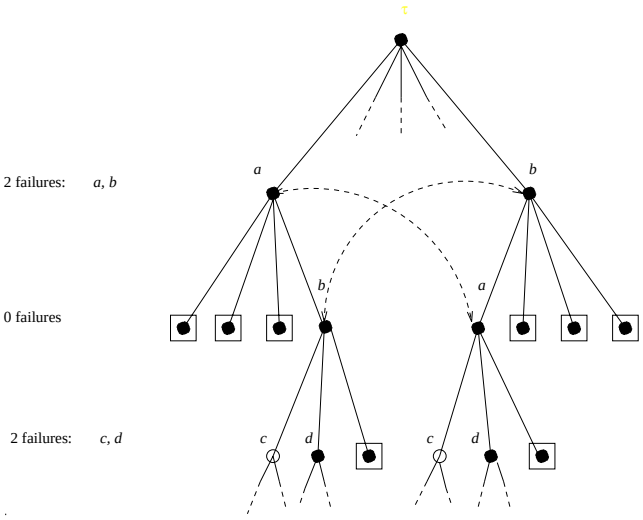


FIG. 3. *Cross corruption.*

7. Cross corruption. As discussed in the last section, in order for the number of universally corrupted nodes to grow significantly without the waste growing at the same time, pairs of faulty processors need to cross corrupt nodes in consecutive rounds. In this section, we perform a careful analysis of the conditions that must be met for cross corruption to succeed. We then use this analysis to fine tune the

[7] Indeed, Berman and Garay [5] have shown that for the resolve function of [2] for $n > 3t$, it is possible to construct an exponential tree this way, despite fault masking, early stopping, and monitor voting.

function F we use for fixing nodes, to limit the number of times that a cross corruption can take place without the waste increasing. This analysis will yield essentially all of the necessary ingredients for our final protocol.

The first property of cross corruption we shall use is the fact that in cross corruption, each universally corrupted node τa in the first of the two rounds has at most one universally corrupted child τab. As a result, if the node τa is not closed in a small number of rounds, then all of its children other than τab must be fixed in all trees. Moreover, since τa is not closed after these nodes become fixed, the number of its children fixed to $\mathtt{par}(|\tau a|)$ and the number fixed to its complement are uniquely determined. More specifically, we have the following lemma.

LEMMA 7.1. *In a* Δ-EIG *protocol with admissible function* F*, let* $v = \mathtt{par}(|\tau a|)$ *and let* τa *be a universally corrupted node whose only universally corrupted child is* τab*. For every nonfaulty processor* i*, if* τa *is not closed in* $\mathtt{tree}_i$ *by the end of round* $|\tau| + 6$*, then all of its children other than* τab *are fixed in* $\mathtt{tree}_i$ *at that point. Moreover, let* A *denote the set of processors* x *for which the node* τax *is fixed to* v *in* $\mathtt{tree}_i$*, and let* B *denote the set of processors* y *such that* τay *is fixed to* $1 - v$*. Then* $\#A = F(|\tau a|) - 1$ *and* $\#B = n - |\tau a| - F(|\tau a|)$.

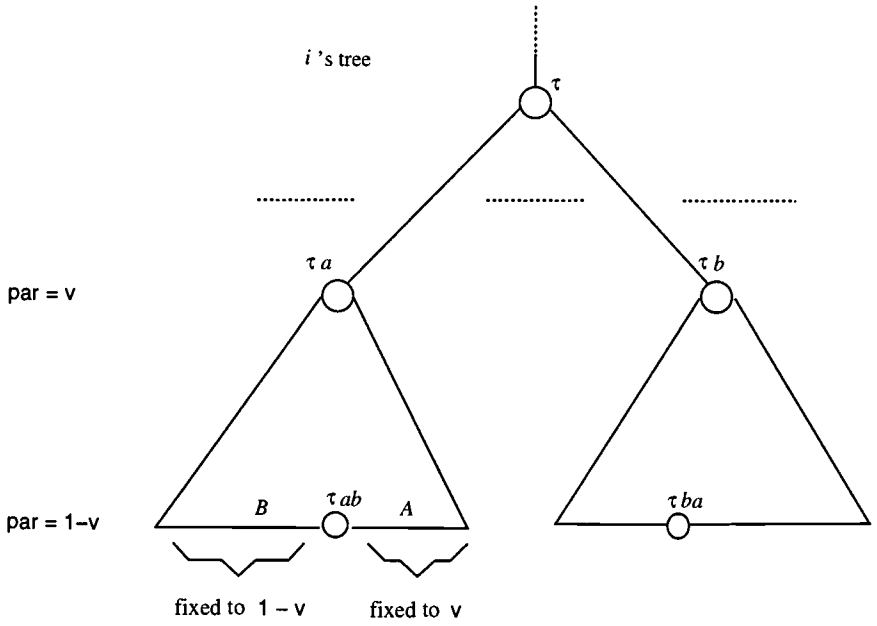


FIG. 4. *Two universally corrupted nodes.*

Proof. The scenario described in the statement of the lemma is depicted in Figure 4. Corollary 6.17 implies that a node τax that is not universally corrupted must be closed in all trees by the end of round $|\tau ax| + 4 = |\tau| + 6$. Since we are assuming that τa is not closed in $\mathtt{tree}_i$ at that point, it follows that every such node τax will necessarily be fixed in $\mathtt{tree}_i$ by the end of round $|\tau| + 6$. Since τab is the only universally corrupted child of τa, we obtain that all other children of τa must be fixed in $\mathtt{tree}_i$ by the end of round $|\tau| + 6$. Let A be set of processors x such that the node τax is fixed to v in $\mathtt{tree}_i$ (by the end of round $|\tau| + 6$), and let B the set of processors y such that the node τay is fixed to $1 - v$. This is depicted in Figure 4.

Since τa is not closed in $\texttt{tree}_i$ by the end of round $|\tau|+6$ we are guaranteed that $\#A < F(|\tau a|)$ and $\#B < n - |\tau a| + 1 - F(|\tau a|)$. However, the fact that only one of τa's children is universally corrupted implies that $\#A + \#B = n - |\tau a| - 1$. It follows that $\#A = F(|\tau a|) - 1$ and $\#B = n - |\tau a| - F(|\tau a|)$. □

Another observation regarding cross corruption is the following. Recall that every node of the safe subtree must be publicly committed either to 0 or to 1. If a node τac is publicly committed to v and then becomes fixed to $1-v$, then by rule FD2 everybody will discover that c is faulty by the end of round $|\tau ac|+2 = |\tau|+4$, and c will become disabled by then. As a result, we obtain the following lemma.

LEMMA 7.2. *Under the conditions and notation of Lemma* 7.1*, let $C \subseteq B$ consist of the processors c such that τac is publicly committed to v. Then $C \subseteq (\mathcal{D}(|\tau|+5) \setminus \mathcal{D}(|\tau|+2))$.*

Proof. By definition of C, for every nonfaulty processor j we have that each node τac with $c \in C$ is committed to v in $\texttt{tree}_j$. In addition, since $C \subseteq B$, we have that τac is fixed to $\texttt{right}^F(\tau ac) = 1 - v$ in $\texttt{tree}_i$ by the end of round $|\tau| + 6$. As a result, we have by Theorem 6.8 and Lemma 6.7 that τac cannot become fixed to v in $\texttt{tree}_j$. Moreover, since, by assumption, τa is not closed in $\texttt{tree}_i$ by the end of round $|\tau| + 6$, we have by Corollary 6.16 that τa is not closed in $\texttt{tree}_j$ by the end of round $|\tau| + 4 = |\tau ac| + 2$. Hence, τac can be closed in $\texttt{tree}_j$ at the end of round $|\tau ac|+2$ only if it is fixed (to $1-v$) in $\texttt{tree}_j$ at that point. If it is indeed fixed to $1 - v$ in $\texttt{tree}_j$ at the end of round $|\tau| + 4$, then τac is committed at that point to $1 - v$ in $\texttt{tree}_j$ by Lemma 6.12. If not, then it is committed to $1 - v$ at that point by C3. Since τa is not closed in $\texttt{tree}_j$ by the end of round $|\tau| + 4|$, we have that τac is not closed in $\texttt{tree}_j$ by the end of round $|\tau ac|$. Thus, in either case, rule FD2 implies that j detects c as faulty by the end of round $|\tau| + 4$. Since this argument applies for every nonfaulty processor j, we have that c is disabled at the end of round $|\tau| + 4$. Notice, however, that c could not have been disabled at the end of round $|\tau| + 2$, for if it had been disabled at that point, then, by Lemma 6.2, τac would not be publicly committed to v: all nonfaulty processors would be reporting $1 - v$ for τac by masking c. By definition of B, this contradicts the assumption that $c \in C \subseteq B$. We conclude that $C \subseteq (\mathcal{D}(|\tau| + 4) \setminus \mathcal{D}(|\tau| + 2))$. Since $\mathcal{D}(|\tau| + 5) \supseteq \mathcal{D}(|\tau| + 4)$, we obtain that $C \subseteq (\mathcal{D}(|\tau| + 5) \setminus \mathcal{D}(|\tau| + 2))$ and we are done. □

Lemma 7.2 implies that, in order to avoid having the waste grow in an instance of cross corruption, the size of the set C must remain small. Specifically, as we shall see later on, it is necessary that $\#C \leq 2$. When this is the case, then the vast majority of nodes τay with $y \in B$ must be publicly committed to $1 - v$. As a result, if B is large enough, we can use the fault-detection rule FD3 to force the processors in A to mask a in round $|\tau| + 3$. More specifically, we have the following lemma.

LEMMA 7.3. *Under the conditions and notation of Lemma* 7.1*, let $B' \subseteq B$ consist of the processors $y' \in B$ such that the node $\tau ay'$ is publicly committed to $1-v$. Assume $\#B' \geq 2(t+1-|\tau a|)+1$. If the node τba is not closed in $\texttt{tree}_i$ at the end of round $|\tau|+7$, then every processor $x \in A$ for which τbax is not fixed to $1-v$ in $\texttt{tree}_i$ by the end of round $|\tau| + 7$ is disabled by the end of round $|\tau| + 5$.*

Proof. Consider the following two cases.

(i) τbax is not closed in $\texttt{tree}_i$ by the end of round $|\tau| + 7$: In this case, by Corollary 6.17 we have that τbax is universally corrupted, and by Corollary 6.19 we obtain that x is disabled by the end of round $|\tau bax| + 2 = |\tau| + 5$.

(ii) τbax is closed in $\texttt{tree}_i$ by the end of round $|\tau| + 7$: Since, by assumption, τba is not closed in $\texttt{tree}_i$ by the end of round $|\tau|+7$, if τbax is closed in $\texttt{tree}_i$ at that

point, then it must be fixed in $\texttt{tree}_i$. Moreover, since we have assumed that τbax is not fixed to $1-v$, it must be fixed to v in $\texttt{tree}_i$ by the end of round $|\tau|+7$. As a result, for every nonfaulty processor j, the node τbax does not become fixed to $1-v$ in $\texttt{tree}_j$. We now show that every nonfaulty processor j must have detected x as faulty no later than in round $|\tau|+5$.

Let j be an arbitrary nonfaulty processor. Recall that every node τax with $x \in A$ is fixed in $\texttt{tree}_i$ to $v \neq \texttt{par}(|\tau ax|)$ by the end of round $|\tau|+6$. By Lemma 6.13 we thus have that every node τax with $x \in A$ is publicly committed to v. If x does not send j a $\texttt{mask}(x,a)$ message by the end of round $|\tau|+3$ then the conditions of the fault detection rule FD3 hold for j with respect to x:

(i) τax is committed to v in $\texttt{tree}_j$; and

(ii) since all nodes τay with $y \in B'$ are publicly committed to $1-v$, at least $\#B' \geq 2(t+1-|\tau a|)+1$ such nodes τay are committed to $1-v$ in $\texttt{tree}_j$ by the end of round $|\tau ax|+1$. It follows that j will detect x as faulty in round $|\tau|+3$.

If $x \in \mathcal{F}_j(|\tau|+3)$ then we are done. Otherwise, x sent j a $\texttt{mask}(x,a)$ message by the end of round $|\tau|+3$, and hence by the masking rule we have that $\texttt{tree}_j(\tau bax) = 1-v$, so that τbax is committed to $1-v$ in $\texttt{tree}_j$ by C1. In addition, since

(i) τbax is not closed in $\texttt{tree}_j$ by the end of round $|\tau bax|+1$; and

(ii) τbax is not fixed to $1-v$ by the end of round $|\tau bax|+2 = |\tau|+5$,

we have that τbax is committed in $\texttt{tree}_j$ to v by C3 by the end of round $|\tau|+5$. Given that τba is not closed in $\texttt{tree}_i$ by the end of round $|\tau|+7$, Corollary 6.16 implies that τba is not closed in $\texttt{tree}_j$ by the end of round $|\tau|+5$. It follows that τbax is not closed in $\texttt{tree}_j$ by the end of round $|\tau bax|+2 = |\tau|+5$. Hence, by fault discovery rule FD2, we obtain that $x \in \mathcal{F}_j(|\tau|+5)$ and we are done. □

Roughly speaking, Lemma 7.3 implies that in a successful cross corruption, most members of A must mask a when reporting on τba. The ones that do not will become disabled by the end of round $|\tau|+5$. As a result, if $F(|\tau|) \geq F(|\tau|+1)+3$, then for τba not to close by the end of round $|\tau|+7$, at least three members of A must become disabled by the end of round $|\tau|+5$. As we shall see, this will be sufficient to guarantee that a cross corruption in these rounds must increase the waste by at least 1.

7.1. A concrete admissible function. Lemmas 7.2 and 7.3 motivate us to seek an admissible function F with the following two properties:

1. $F(r) \leq t+r-4$ for all $2 \leq r \leq t$, and
2. $F(r) \geq F(r+1)+3$ for all $2 \leq r \leq t$.

In the notation of the above lemmas, the first property guarantees, for nodes τ satisfying $|\tau| \geq 2$, that if $\#B' \leq 2(t+1-|\tau a|)$ then $\#C \geq 3$. As we shall see, a combination of both properties implies that a cross corruption must necessarily cause an increase in the waste. Unfortunately, an admissible function with both properties does not exist. We now turn to define a function that will have the first property, and will approximate the second property. This property will hold for all but a small number of rounds r.

Recall that, for F to be admissible, it must satisfy $t+r-1 \geq F(r) \geq t-r+2$ for $2 \leq r \leq t$. Intuitively, we divide the execution into "phases" consisting of many rounds each. In each phase we start with $F(r)$ being close to $t+r$, and we reduce the threshold by steps of 3 from one round to the next until we come close to $t-r$. A new phase then begins. Rather than at $t+r-1$, a phase will start with the threshold no greater that $t+r-4$, to guarantee the first property described above. Thus, both desired properties are maintained during a phase, but they are violated in the

transition between phases. Luckily, the number of such transitions will be shown to be logarithmic in n.

We define $\mathsf{rem}(k)$ to be the difference between k and the largest power of 2 that is smaller than k. More precisely stated,

$$\mathsf{rem}(k) \stackrel{\text{def}}{=} k - 2^{\lfloor \log_2 k \rfloor}.$$

Notice that $\mathsf{rem}(k) = 0$ for $k \geq 1$ precisely if k is a power of 2. Moreover, one property of rem that we shall use in the sequel is that, for all natural numbers $k \geq 1$ we have $0 \leq \mathsf{rem}(k) \leq \frac{k+1}{2} - 1$, so that, in particular, we have $0 \leq 4\mathsf{rem}(k) \leq 2k - 2$.

Our threshold function F^* is defined as follows:

$$F^*(r) = \begin{cases} t+1 & \text{for } 0 \leq r \leq 4; \text{ and} \\ t + r - 4 - 4\mathsf{rem}(r-3) & \text{for } r \geq 5. \end{cases}$$

A few essential properties of the function F^* that we shall find useful in the sequel are as follows.

LEMMA 7.4. *If $r \geq 5$ then*

$$t + r - 4 \;\geq\; F^*(r) \;\geq\; t - r + 4.$$

Proof. Since $0 \leq 4\mathsf{rem}(k) \leq 2k-2$ for $k \geq 2$, we have that $0 \leq 4\mathsf{rem}(r-3) \leq 2r-8$ for $r \geq 5$. It follows that $t+r-4 \geq F^*(r) \geq t+r-4-(2r-8) = t+r-4-2r+8 = t-r+4$ for $t+1 \geq r \geq 5$ and we are done. □

LEMMA 7.5. *The function F^* is an admissible resolve function.*

Proof. Recall that a function F is admissible if it satisfies $F(0) = F(1) = t+1$ and $t + r - 1 \geq F(r) \geq t - r + 2$ for $r \geq 2$. For $r = 0, 1$ we have $F^*(r) = t+1$ as desired. For $2 \leq r \leq 4$ we have $F^*(r) = t+1$ and $t + r - 1 \geq t + 1 \geq t - r + 2$. For $r \geq 5$, Lemma 7.4 states that $t + r - 4 \geq F^*(r) \geq t - r + 4$. We thus have $t + r - 1 \geq t + r - 4 \geq F^*(r) \geq t - r + 4 \geq t - r + 2$ and we are done. □

LEMMA 7.6. *Let $r \geq 5$ and assume that $r - 2$ is not a power of 2. Then $F^*(r) - F^*(r+1) = 3$.*

Proof. Since $r \geq 5$, we have that $F^*(r) = t + r - 4 - 4\mathsf{rem}(r-3)$, while $F^*(r+1) = t + r - 3 - 4\mathsf{rem}(r-2)$. The fact that $r-2$ is not a power of 2 implies that $\mathsf{rem}(r-2) = \mathsf{rem}(r-3)+1$. It follows that $F^*(r) = t+r-3+1-4(\mathsf{rem}(r-2)-1) = F^*(r+1) - 1 + 4 = F^*(r+1) + 3$ and we are done. □

Obviously, the number of rounds $r \leq t$ for which $r-2$ is a power of 2 is roughly $\log_2 t$.

7.2. The main lemma. We are now in a position to prove that, given our fixing and fault detection rules, the price of cross corruption is high. In other words, that for cross corruption to take place, many faulty processors must become disabled. As a result, cross corruption will no longer be a problem for monitor voting. We now have the following.

LEMMA 7.7. *Let τa and τb be universally corrupted nodes, such that*

(i) *$|\tau| \geq 4$ and $|\tau| - 1$ is not a power of 2;*

(ii) *the only universally corrupted child of τa is τab and the only universally corrupted child of τb is τba; and*

(iii) *for some i, both of the nodes τa and τb are not closed in $\mathtt{tree}_i$ by the end of round $|\tau| + 7$.*

Then

$$\#(\mathcal{D}(|\tau|+5) \setminus \mathcal{D}(|\tau|+2)) \ \geq \ 5.$$

Proof. The situation considered in this lemma is partly illustrated by Figure 4. We first prove the following about a and b.

CLAIM 7.8. $a, b \in (\mathcal{D}(|\tau|+5) \setminus \mathcal{D}(|\tau|+2))$.

Proof. The fact that τa and τb are not closed in $\texttt{tree}_i$ by the end of round $|\tau|+7$ implies, by Corollary 6.17, that these nodes are universally corrupted. By Corollary 6.19 we thus obtain that both a and b must be disabled no later than at the end of round $|\tau a|+2 = |\tau|+3$. However, if either of them were disabled by the end of round $|\tau|+2$, then all nonfaulty processors would be masking them in round $|\tau|+3$, and by Lemma 6.2 we would have that τba and τab would not be corrupted in any nonfaulty processor's tree, contradicting our assumption about a and b. It follows that

$$a, b \in (\mathcal{D}(|\tau|+3) \setminus \mathcal{D}(|\tau|+2)) \subseteq (\mathcal{D}(|\tau|+5) \setminus \mathcal{D}(|\tau|+2)). \qquad \square$$

Given Claim 7.8, we need to show that, under the conditions of the lemma, at least three additional faulty processors must become disabled between rounds $|\tau|+3$ and $|\tau|+5$.

We shall perform our analysis based on the subtree rooted at τa. Assume the conditions of the lemma hold, and without loss of generality let $\texttt{par}(|\tau a|) = v$. Since τab is the only universally corrupted child of τa, it follows by Corollary 6.17 that all other children of τa must be fixed in $\texttt{tree}_i$ by the end of round $|\tau|+6$. Let A denote the set of processors x such that the node τax is fixed to v in $\texttt{tree}_i$ (by the end of round $|\tau|+6$), and let B the set of processors y such that the node τay is fixed to $1-v$. This is depicted in Figure 4. The conditions of Lemma 7.1 are satisfied, and as a result we have that $\#A = F^*(|\tau a|) - 1$ and $\#B = n - |\tau a| - F^*(|\tau a|)$.

As in Lemma 7.2, let C denote the subset of B consisting of processors c such that τac is publicly committed to v.

Let us denote $B' \stackrel{\text{def}}{=} B \setminus C$. By definition, every node $\tau ay'$ with $y' \in B'$ is publicly committed to $1-v$, and is hence committed to $1-v$ in every nonfaulty processor's tree. We now show that, if B' is "small" (namely, $\#B' \leq 2(t+1-|\tau a|)$), then C is large enough (i.e., $\#C \geq 3$) to yield the lemma. If B' is not small, however, we shall use Lemma 7.3 to show that sufficiently many processors, this time members of A, must be disabled. Thus, either way we obtain that at least three processors in addition to a and b become disabled in rounds $|\tau|+3$ through $|\tau|+5$.

As described above, we consider two cases:

(i) $\#B' \leq 2(t+1-|\tau a|)$: In this case, we claim that $\#C \geq 3$. This follows from the fact that $\#B = n - |\tau a| - F^*(|\tau a|)$ and $\#C = \#B - \#B'$. The calculation is as follows:

$$\begin{aligned}
\#C &\geq n - |\tau a| - F^*(|\tau a|) - 2(t+1-|\tau a|) \\
&= n - 2t + |\tau a| - 2 - F^*(|\tau a|) \\
&\geq n - 2t + |\tau a| - 2 - (t + |\tau a| - 4) \\
&= n - (3t+1) + 3 \geq 3.
\end{aligned}$$

Lemma 7.2 implies that $C \subseteq (\mathcal{D}(|\tau|+5) \setminus \mathcal{D}(|\tau|+2))$. In the claim above we showed that $a, b \in (\mathcal{D}(|\tau|+5) \setminus \mathcal{D}(|\tau|+2))$. Since $a \notin C$ and $b \notin C$, we obtain that $\#(\mathcal{D}(|\tau|+5) \setminus \mathcal{D}(|\tau|+2) \geq 5$.

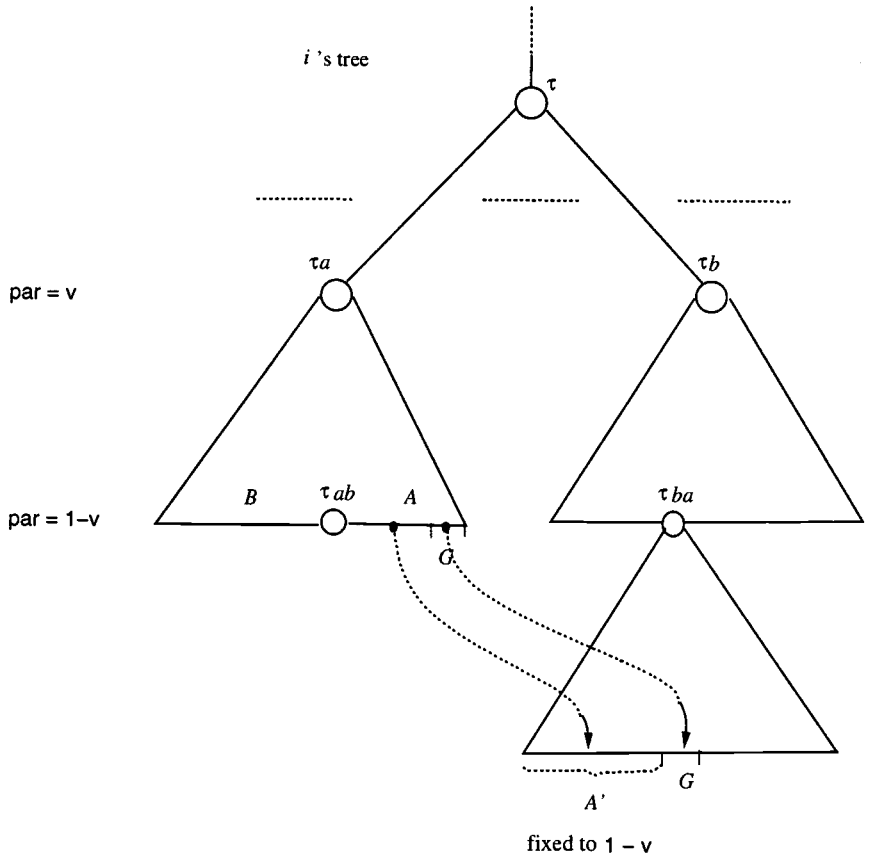


FIG. 5. *A set G of newly disabled processors.*

(ii) $\#B' \geq 2(t+1-|\tau a|)+1$: Since we are assuming that $|\tau| \geq 4$ we have $|\tau a| \geq 5$. By Lemma 7.6 we thus have that $F^*(|\tau a|) - F^*(|\tau ba|) = 3$. Let A' consist of the processors x' such that $\tau bax'$ is fixed to $1-v$ in $\texttt{tree}_i$ by the end of round $|\tau|+7$. Since τb is not closed in $\texttt{tree}_i$ by the end of round $|\tau|+7$ and τba is the only universally corrupted child of τb, we have that τba is also not closed in $\texttt{tree}_i$ at that point. It follows that $\#A' \leq F^*(|\tau ba|) - 1 = F^*(|\tau a|) - 4 = \#A - 3$. Define $G \stackrel{\text{def}}{=} (A \setminus A')$ (see Figure 5). In particular, we obtain that $\#G \geq 3$. Moreover, if a processor $z \in A$ is disabled by the end of round $|\tau|+3$, then Lemma 6.2 and the masking rules imply that τbaz is fixed in $\texttt{tree}_i$ to $\texttt{par}(|\tau ba|) = 1-v$ by the end of round $|\tau|+4$. It follows that no processor $x \in G$ is disabled by the end of round $|\tau|+3$. The conditions of Lemma 7.3 are satisfied with respect to every $x \in G$, and hence by Lemma 7.3 we have that $G \subseteq (\mathcal{D}(|\tau|+5) \setminus \mathcal{D}(|\tau|+2))$. Notice that $a, b \notin G$. Thus, we again obtain that $\#(\mathcal{D}(|\tau|+5) \setminus \mathcal{D}(|\tau|+2)) \geq 5$, and we are done. □

As a consequence of Lemma 7.7 and the definition of `Waste` we obtain the following.

COROLLARY 7.9. *If the conditions of Lemma* 7.7 *hold with respect to τ, then* $\texttt{Waste}(|\tau|+5) > \texttt{Waste}(|\tau|+1)$.

Proof. We have defined $\texttt{Waste}(r)$ by:

$$\texttt{Waste}(r) \;=\; \#\mathcal{D}(r) - r + \texttt{deficit}(r) - \texttt{correction}(r).$$

By definition, we have that $\texttt{correction}(r) \geq 0$, and $\texttt{deficit}(r) - \texttt{correction}(r) \geq 0$. It thus follows that

$$\#\mathcal{D}(r) + \texttt{deficit}(r) \geq \texttt{Waste}(r) + r \geq \#\mathcal{D}(r). \tag{1}$$

In particular, we have that $\texttt{Waste}(|\tau|+5) + |\tau| + 5 \geq \#\mathcal{D}(|\tau|+5)$. Recall that the deficit at $|\tau|+1$ corresponds to processors that have universally corrupted nodes at depth $|\tau|$ but are not yet disabled at time $|\tau|+1$. All of these processors are discovered as faulty by all nonfaulty processors, and are hence disabled, by time $|\tau|+2$. Thus, we have that $\#\mathcal{D}(|\tau|+2) \geq \#\mathcal{D}(|\tau|+1) + \texttt{deficit}(|\tau|+1)$. By equation (1) above we thus obtain that $\#\mathcal{D}(|\tau|+2) \geq \texttt{Waste}(|\tau|+1) + |\tau| + 1$.

By Lemma 7.7 we have that $\#(\mathcal{D}(|\tau|+5) \setminus \mathcal{D}(|\tau|+2)) \;\geq\; 5$. Since the set $\mathcal{D}$ grows monotonically, $\mathcal{D}(|\tau|+5) \supseteq \mathcal{D}(|\tau|+2)$, and we can therefore conclude that $\#\mathcal{D}(|\tau|+5) \geq \#\mathcal{D}(|\tau|+2) + 5$. In summary, we have

$$\texttt{Waste}(|\tau|+5) + |\tau| + 5 \geq \#\mathcal{D}(|\tau|+5) \geq \#\mathcal{D}(|\tau|+2) + 5 \geq \texttt{Waste}(|\tau|+1) + |\tau| + 1 + 5.$$

Deducting $|\tau|+5$ from both sides, we obtain that $\texttt{Waste}(|\tau|+5) \geq \texttt{Waste}(|\tau|+1) + 1$, so that $\texttt{Waste}(|\tau|+5) > \texttt{Waste}(|\tau|+1)$ and we are done. □

Lemma 7.7 implies that from round five on, the only rounds in which cross corruption can take place without increasing the waste of the run are pairs of rounds $r, r+1$ such that $r-1$ is a power of 2. In particular, since $r \leq t$, this can happen no more than $\log_2 t + O(1)$ times. These rounds can increase the number of universally corrupted nodes at any given depth in the tree by a factor of at most $O(t)$.

7.3. Early stopping in Δ-agreement. A crucial property of the Δ-EIG protocol is captured by Corollary 6.16. It states that two rounds after a node σ is closed in one nonfaulty processor's tree, it will be closed in all processors' trees. Obviously, once σ is closed in $\texttt{tree}_i$, processor i has no use for the descendants of σ. Nevertheless, it might still need to record values and perform fault detection, in order to continue reporting on nodes in order to allow σ to close in the trees of other processors. What Corollary 6.16 implies, then, is that i needs to relay values in the subtree rooted at σ for at most two rounds after σ is closed in $\texttt{tree}_i$. This suggests that we can modify the Δ-EIG protocol to obtain an early-stopping protocol as follows.

(a) Rather than reporting on all internal nodes in the Δ-EIG tree, processor i will report on a node σ in round $|\sigma|+1$ only if σ was not closed in $\texttt{tree}_i$ by the end of round $|\sigma|-2$. (Recall that a node can be closed before any value is stored in it; all that is needed for it to be closed is that one of its ancestors should be fixed to a value.) To implement this rule, all that is needed is for i to handle and report only on the children and grandchildren of nodes that fix by rule Fx2.

(b) We modify the halting condition for a processor i to:

- *Halting′:* Processor i continues to record information, perform fault detection, and report on values for two rounds after the root λ is closed in $\texttt{tree}_i$. At the end of these two rounds (and no later than at the end of round $t+1-\Delta$), it halts.

(c) There are a couple of details we need to take care of once we modify the protocol as described above. They result from the fact that it is now possible not to receive a message from a *nonfaulty* processor. This can only happen, however, in cases in which these values are of no use to the receiver. This leads to modifications

of the recording and masking rule, and to a modification of the definition of an ill-formatted message, which in turn affects the process of fault detection. We start by describing the latter. We shall extend the notion of an ill-formatted message as follows: If node σ is not closed in $\texttt{tree}_i$ by the end of round $|\sigma|$, and processor j's message in round $|\sigma|+1$ does not report values for all children of σ, then the message is considered ill formatted and i will detect j as faulty by rule FD0. One consequence of this definition is that if j sends no message to i in round $r+1$ and the root λ is not closed in $\texttt{tree}_i$ by the end of round r, then i detects j as being faulty.

Finally, the recording and masking rule is modified for the case in which the root λ is fixed in $\texttt{tree}_i$ at the end of round r and i receives no message in round $r+1$ from some processor $j \notin \mathcal{F}_i(r)$. Since i needs to continue to report on values in round $r+2$ by the halting$'$ rule, it acts as follows. For nodes that correspond to processors z for which j has issued $\texttt{mask}(j,z)$ reports in the first r rounds, there is no problem. For all other nodes, we choose to have i consider j as reporting the same values that i has reported, for all depth r nodes that i reports on in round $r+1$. This is related to the reconstruction method advanced by Zamsky [26, 28], and by Berman, Garay, and Perry [7]. One feature of this choice is that it keeps the fault detection rule FD2 from ever causing i to mistakenly "detect" j as faulty. (While such a mistaken detection would not change i's decision in the agreement process being executed, it becomes problematic when we run a number of agreement processes in parallel, and use a common fault-detection module as will be described in section 8.1.)

We call the resulting protocol the Δ-ES protocol (the ES stands for *early stopping*). Despite possibly reporting on much fewer nodes in a run of Δ-ES than in similar runs of Δ-EIG, the processors' behavior in Δ-ES maintains an important invariant: If a node σ is not closed in $\texttt{tree}_i$ at the end of round $|\sigma|$ and none of σ's ancestors is closed in $\texttt{tree}_i$ by the end of round $|\sigma|+1$, then all nonfaulty processors report a value for σ in round $|\sigma|+1$, as well as performing fault detection for all of σ's children in round $|\sigma|+1$, and reporting on them in round $|\sigma|+2$. This ensures the following:

(i) For every nonfaulty i and j, the first node to close in $\texttt{tree}_j$ along any path from the root, is guaranteed to close in $\texttt{tree}_i$ at most two rounds after it does in $\texttt{tree}_j$. Hence, the information that processor i would receive in Δ-EIG but does not receive in Δ-ES does not affect i's decision.

(ii) The commitment rules operate in Δ-ES as they do in Δ-EIG, since they depend only on the values stored in a node (C1), its children (C2), and grandchildren (C3). The fault detection rule FD2 remains sound, since it depends solely on these commitment rules.

(iii) The soundness of fault detection rule FD3 is maintained: If the node τaz is not closed in $\texttt{tree}_i$ by the end of round $|\tau az|+1$, then all processors are guaranteed to store the children of τa in the previous round, perform failure detection at the end of round $|\tau az|$, and report on nodes in round $|\tau az|+1$. Hence, a processor that does not mask the culprit processor a of FD3 in the designated round, must be faulty.

As a result, all of the properties proven for Δ-EIG in the previous sections hold once we move to the Δ-ES protocol in the following proposition.

PROPOSITION 7.10. *All of the statements from Lemma* 6.7 *through Lemma* 7.7 *hold for the* Δ*-ES protocol.*

The main aim of the protocol Δ-ES is to allow us to refrain from having to construct exponential-size EIG trees and hence from having to send an exponential amount of communication. At this point, we are able to prove a polynomial relation

between the number of universally corrupted nodes and the size of trees. This will reduce our problem to keeping the number of universally corrupted nodes polynomial, which will be done in the later sections. We thus have the following corollary.

COROLLARY 7.11. *Assume that the last nonfaulty processor halts by the end of round r, and let the number of universally corrupted nodes σ of depth $|\sigma| \leq r-6$ be T. Then, for every nonfaulty processor i, the total number of nodes in $\texttt{tree}_i$ is bounded by $O(n^6 T)$.*

Proof. Let σ be a universally corrupted node. The node σ has $O(n)$ children σj that are not universally corrupted. Corollary 6.17 implies that each such child is closed in $\texttt{tree}_i$ by the end of round $|\sigma j|+4$. It follows that σj has at most $O(n^4)$ descendants in $\texttt{tree}_i$ by the time it is closed. By definition of the Δ-ES protocol, processor i will need to store one level of nodes, beyond these $O(n^4)$ nodes, in the subtree rooted in σj. It follows that σ can have at most $O(n^6)$ descendants in $\texttt{tree}_i$ that are not themselves descendants of a universally corrupted child of σ. By accounting each node of $\texttt{tree}_i$ to its closest ancestor that is universally corrupted, the claim follows. □

8. The `Sliding-flip` protocol. We now have all of the ingredients necessary to define the final combined protocol. Intuitively, the protocol will be a variant of monitor agreement in which the monitors will be instances of Δ-agreement. We remark that in such a setting, there is a distinction between the *global* round number, which is counted from the start of the original agreement process, and the *local* round number of a specific instance of Δ-ES, which is counted from the start of this instance. The *global depth* and *local depth* of a node are defined analogously. The definition of Δ-ES and our analysis of Δ-ES use local round numbers. In our protocol, one instance of Δ-agreement is spawned in every round, for rounds $1 \leq r \leq t$. In round r, the agreement process that is generated will be a Δ-ES for $\Delta = r-1$. Since, by the Halting′ property, such an instance will complete in $t+1-\Delta = t+2-r$ rounds, we obtain that every such process will be complete by the end of (global) round $r-1+t+2-r = t+1$. We call such a process *$(t+1)$-bounded*. Before we provide the details of the voting rules and properties of these instances of Δ-agreement, we now describe a general method by which such agreement processes can be combined. This will be used when we come to combine the agreement processes in the final description of the combined protocol.

8.1. Preempt-on-one. In our combined protocol, we initiate one Δ-agreement process $\texttt{M}^r$ in every round r, where the parameter $\Delta = r-1$ is used in $\texttt{M}^r$. In this section we describe the method by which we combine the different agreement processes. This method is stated in slightly more general terms than we need, as it applies elsewhere as well. We call a set of agreement processes being executed concurrently with respect to a single fault detection module (FDM) an *ensemble*, and denote ensembles by $\mathcal{E}$. Different agreement processes are said to use a single FDM if for all nonfaulty processor i and (global) round r, the set $\mathcal{F}_i(r)$ used by one agreement process is the same as that used by the others.

We now define an operation that takes an ensemble and turns it into a single protocol. In executing $\texttt{Preempt-on-One}(\mathcal{E})$, a nonfaulty protocol concurrently executes all of the processes in $\mathcal{E}$. It decides and halts according to the following rule:

`Dec0`. Processor i decides 0 on $\texttt{Preempt-on-One}(\mathcal{E})$ and halts once i has halted with a decision of 0 on all processes in $\mathcal{E}$.

`Dec1`. Processor i decides 1 on $\texttt{Preempt-on-One}(\mathcal{E})$ and halts (preempting all agreement processes underway) once i has halted with a decision of 1 on

some process in $\mathcal{E}$.

The following property of `Preempt-on-One`$(\cdot)$ is fairly immediate, and will turn out useful.

LEMMA 8.1. *Let $\mathcal{E}$ be an ensemble of agreement processes. If all processes in $\mathcal{E}$ are $(t+1)$-bounded and satisfy the agreement property, then* `Preempt-on-One`$(\mathcal{E})$ *is $(t+1)$-bounded and satisfies the agreement property.*

Proof. We start by showing that `Preempt-on-One`$(\mathcal{E})$ is an agreement protocol. Assume that some processor i decides 0 on `Preempt-on-One`$(\mathcal{E})$ in a given run. It follows from `Dec0` that i decides 0 on all processes in $\mathcal{E}$ in this run. Since the instances all satisfy the agreement property, no other nonfaulty processor j will decide 1 on any of the processes in $\mathcal{E}$. In addition, since the processes in $\mathcal{E}$ are all $(t+1)$-bounded, it follows that j will actually decide 0 on every process in $\mathcal{E}$ no later than in round $t+1$. We conclude that every nonfaulty processor will halt in `Preempt-on-One`$(\mathcal{E})$ with a decision of 0 by the end of round $t+1$. Assume now that i decides 1 on `Preempt-on-One`$(\mathcal{E})$ in a given run. Let $\mathtt{M} \in \mathcal{E}$ be the process that triggers i's decision. In particular, it follows that processor i decides 1 and then halts on `M` at the end of some round $r_i \leq t+1$. Let $j \neq i$ be any other nonfaulty processor. There are two possibilities: (i) Processor j decides and halts on `Preempt-on-One`$(\mathcal{E})$ before it has a chance to decide and halt on the process `M`. By rule `Dec0` in the definition of `Preempt-on-One`$(\mathcal{E})$, a decision of 0 can only be taken after j has decided and halted on all processes in $\mathcal{E}$. It follows that j could only have decided 1, so its decision is in agreement with processor i's decision. Moreover, since `M` was $(t+1)$-bounded, and j decided before it had a chance to decide on `M`, we obtain that j decides before round $t+1$. (ii) The other possibility is that j does manage to decide and halt on `M`. Here again because `M` satisfies the agreement property, j will decide 1 on `Preempt-on-One`$(\mathcal{E})$. In addition, since `M` was $(t+1)$-bounded, j decides no later than in round $t+1$. In summary, we have that in all cases, `Preempt-on-One`$(\mathcal{E})$ satisfies the agreement property and is $(t+1)$-bounded, and we are done. □

8.2. Putting it all together. As described above, the ensemble generated in the combined protocol consists of instances of Δ-ES protocols, initiated one per round, in rounds $1 \leq R \leq t$. We use R in this section to refer to global round numbers. In round R, a monitor process initiated in global round K will have its own local round number $r = R + 1 - K$. All the instances of Δ-ES protocols invoked use the function F^* defined in section 7.1. Notice that the function F^* is applied, for every instance of Δ-ES, relative to the local round count. Thus, in the same global round, each agreement process has its own local round number. The combined protocol will be `Preempt-on-One`$(\mathcal{E})$, for the ensemble thus generated. We now describe how to determine a processor's initial vote in any given monitor process.

Clearly, in round $R = 1$, the value a processor i uses as its initial value is its original initial value v_i. In later rounds $R > 1$, the initial value of processor i in monitor process $\mathtt{M}^R$, is essentially the same as the one we mentioned in section 4.3 for $n > 4t$. For the purpose of this voting rule, we say that processor i has *detected* another processor z *as being disabled* by the end of round K if i has received `mask`(j, z) reports from at least $2t + 1$ processors j by that point. The following lemma justifies this terminology.

LEMMA 8.2. *If i has detected z as being disabled by the end of round K, then z is disabled at the end of round K.*

Proof. By definition, i can detect z as being disabled only once i has received at least $2t+1$ `mask`(j, z) reports. At least $t+1$ of these reports are from nonfaulty proces-

sors j. The sending rule implies that a nonfaulty processor sends identical messages to all processors in every round. It thus follows that every nonfaulty processor i' must have received at least $t+1$ $\texttt{mask}(j,z)$ reports by the end of round K. As a result, the fault detection rule FD1 implies that each such processor i' must detect z as being faulty by the end of round K. It follows that z is disabled at that point, and we are done. □

The voting rule on a monitor $\texttt{M}^R$ with $R > 1$ is:

Monitor vote: Processor i will vote 1 on $\texttt{M}^R$ if

(i) at least one node at (global) depth $R-1$ in one of i's trees is not closed; and

(ii) i has detected at least $R-1$ faulty processors as being disabled by the end of round $R-1$.

It will vote 0 on $\texttt{M}^R$, otherwise.

We define the `Sliding-flip` protocol to be the protocol that results from performing `Preempt-on-One` on the ensemble consisting of the original agreement tree together with the monitor agreement processes $\texttt{M}^R$, for $2 \leq R \leq t$, where the votes of the nonfaulty processes are obtained according to the monitor-vote rule above. These agreements processes are all instances of Δ-ES, based on the threshold function F^* defined in section 7.1. For ease of exposition, we shall consider the original agreement process to be $\texttt{M}^1$.

The role of part (i) in the monitor-vote rule is to guarantee that if all initial values are 0, then a decision of 0 will be reached. The role of part (ii) is to guarantee that the monitors satisfy the initial conditions of Δ-agreement: A nonfaulty process votes 1 on a monitor agreement process only if a sufficient number of processors are disabled. In fact, an immediate consequence of Lemma 8.2 is the following corollary.

COROLLARY 8.3. *If at least one nonfaulty processor votes* 1 *in a monitor* $\texttt{M}^{R+1}$*, then* $\#\mathcal{D}(R) \geq R$.

Corollary 8.3 formally shows that the monitors that are initiated in the combined protocol are "legal" instances of Δ-agreement. We can now prove the following proposition.

PROPOSITION 8.4. *The* `Sliding-flip` *protocol is a correct* $(t+1)$*-round Byzantine agreement protocol.*

Proof. Theorem 6.10 and Corollary 8.3 imply that all of the agreement processes initiated in `Sliding-flip` are $(t+1)$-bounded instances of Byzantine agreement. We therefore obtain by Lemma 8.1 that the `Sliding-flip` protocol satisfies the decision and agreement conditions. We need to show that it also satisfies validity. If all initial values are 1, then the root of the initial tree will fix to 1 at the end of round two for all of the nonfaulty processors, so by the definition of `Preempt-on-One` they will all decide 1. Assume that all initial values are 0. In particular, the initial values of all nonfaulty processors are 0. It follows that, for every nonfaulty processor i, at least $2t+1$ of the root's children in the initial tree will store 0 at the end of round one. Since 0 is the preferred value in this round, the root will fix to 0. As a result, all nodes at global depth 1 will be closed for every nonfaulty processor, and by the monitor-vote rule, every such processor i will vote 0 on $\texttt{M}^2$. A straightforward induction on R now shows that all nonfaulty processors vote 0 on all monitors $\texttt{M}^R$, so that all monitors decide 0, and by definition of `Preempt-on-One` every nonfaulty processor i will end up deciding 0 at time $t+1$. □

It remains to show that the protocol is efficient. As a partial converse of Lemma 8.2 we have the following.

LEMMA 8.5. *If z is disabled at the end of round R and no nonfaulty processor has halted by the end of round R, then every nonfaulty processor i will have detected z as being disabled by the end of round $R+1$.*

Proof. If no nonfaulty processor has halted by the end of round R, then they all send messages in round $R+1$. If z is disabled at the end of round R, then every nonfaulty processor j must send a $\texttt{mask}(j,z)$ message to i no later than in round $R+1$. It follows that i is guaranteed to receive at least $2t+1$ such messages by the end of round $R+1$, and we are done. □

Recall that the `Waste` was defined with respect to a given instance of Δ-ES. The parameter r of $\texttt{Waste}(r)$ in a given instance of Δ-ES is a local round number. When we run many instances of Δ-ES concurrently, as in the `Sliding-flip` protocol, we wish to reason about an overall notion of waste. Let us define the global waste at time K, denoted by $\texttt{G\_Waste}(K)$, to be the maximum value of $\texttt{Waste}(K-R)$ in monitor $\texttt{M}^R$, over all $R < K$. Thus, $\texttt{G\_Waste}(K)$ is the maximal value that the waste obtains at global time K. As a consequence of Lemma 8.5 and the monitor voting rule, we now have the following lemma.

LEMMA 8.6. *If $\texttt{G\_Waste}(R) \geq 2$ and some node is universally corrupted, then all nonfaulty processors are guaranteed to halt by the end of round $R+6$.*

Proof. If $R+6 \geq t+1$ the claim is immediate from the definition of the combined protocol, which is guaranteed to halt for every processor by the end of (global) round $t+1$. Assume $R+6 < t+1$. It follows that processors can halt by the end of round $\leq R+6$ only due to their deciding 1 on some monitor. First assume that some nonfaulty processor i halts by the end of round $R+4$, based on deciding 1 on a monitor $\texttt{M}^{R'}$ for some $R' < R+3$ by the end of round $R+2$. It follows from Corollary 6.16 and Proposition 7.10 that all processors will have decided 1 on $\texttt{M}^{R'}$ by the end of round $R+4$ and will halt by the end of round $R+6$. Assume now that no nonfaulty processor has halted by the end of round $R+4$ due to a monitor $\texttt{M}^{R'}$ with $R' < R+3$. The fact that $\texttt{G\_Waste}(R) \geq 2$ implies that $\#\mathcal{D}(R+1) \geq R+2$. Lemma 8.5 states that every nonfaulty processor i detects all members of $\mathcal{D}(R+1)$ as being disabled by the end of round $R+2$. As a result, i will have detected at least $R+2$ disabled processors by the end of round $R+2$. Finally, the fact that some node is universally corrupted implies that, for every nonfaulty processor i, one of i's trees has survived the first round. The monitor voting rule now states that i will vote 1 on $\texttt{M}^{R+3}$. Since all of the processors vote 1 on $\texttt{M}^{R+3}$ and are active until the end of round $R+4$, it follows that they all decide 1 on $\texttt{M}^{R+3}$ at the end of round $R+4$ and will thus halt by the end of round $R+6$. □

Given Lemma 8.6 and Theorem 6.23, we can now prove the following lemma.

LEMMA 8.7. *Choose L such that the last nonfaulty processor halts by the end of round $L+6$. Then, for every nonfaulty i and every monitor $\texttt{M}^R$ with $R \leq L$, the number of universally corrupted nodes in $\texttt{tree}_i$ for $\texttt{M}^R$ is $O(t^2)$.*

Proof. If, for some nonfaulty processor i, none of i's trees survives the first round, then no node is universally corrupted, and we are done. We shall therefore assume from now on that at least one node is universally corrupted. By Lemma 8.6, the waste cannot reach 2 before round L, or all processors would halt before round $L+6$. We thus reason about the rounds preceding L, assuming the waste never reaches 2 in those rounds. We claim that if any node of (global) depth $K \geq 2$ is universally corrupted, then $\texttt{G\_Waste}(K) \geq -2$. Since, for every monitor it is guaranteed by definition that $\texttt{deficit}(r) - \texttt{correction}(r) \geq 0$ for every r, it suffices to show that $\#\mathcal{D}(K) - K \geq -1$. Assume that σ is a node of depth K in a monitor $\texttt{M}^R$ (and hence

$|\sigma| = K-(R-1)$). Since there is a universally corrupted node in $\mathtt{M}^R$, we know that at least one nonfaulty processor voted 1 on this monitor, and hence by the monitor-vote rule and by Lemma 8.2 we have that $\#\mathcal{D}(R-1) - (R-1) \geq 0$. If $|\sigma| \leq 2$ we are clearly done. If, however, $|\sigma| \geq 3$, then the path from the root to σ consists of nodes all of which are universally corrupted. Clearly, by the end of round K, all processors universally corrupting nodes at depths smaller than $K-1$ are disabled. It follows that $|\sigma|-2$ of σ's ancestors contribute fresh disabled processors that were not in $\mathcal{D}(R-1)$. We thus obtain that $\mathcal{D}(K) \geq R-1+|\sigma|-2 = R-1+K-(R-1)-2 = K-2$, and hence $\mathcal{D}(K) - K \geq -2$ and the claim is proven. We conclude that `G_Waste` can vary within a small constant range without causing a monitor to halt the protocol.

It is straightforward to check that if $k \geq 3$ processors universally corrupt nodes at (global) depth R, then $\mathtt{G\_Waste}(R+1) - \mathtt{G\_Waste}(R-1) \geq k-2$. It follows that this can happen only a constant number of times. A round with no universal corruption whatsoever will cause all of the active monitors to close in two rounds. As argued in section 7.2 following Corollary 7.9, the only case in which two processors can corrupt nodes at a given depth in the same tree without increasing the waste is when cross corruption takes place in a pair of rounds $r, r+1$ such that $r-1$ is a power of 2. The number of such rounds $r \leq t+1$ is $\log t + O(1)$. It follows that the number of universally corrupted nodes at any particular level of $\mathtt{tree}_i$ for a monitor $\mathtt{M}^R$ is $O(t)$. Hence, the total number of universally corrupted nodes created by the end of round L in $\mathtt{tree}_i$ for $\mathtt{M}^R$ is bounded by $O(t^2)$. □

As a consequence of Lemma 8.7 and Lemma 7.11 we obtain the following lemma.

LEMMA 8.8. *For every nonfaulty processor i, the total size of each of the t* EIG *trees that i ever constructs is polynomial.*

Finally, as a result of Lemma 8.8 and Corollary 7.11, we obtain the following theorem.

THEOREM 8.9. *The* `Sliding-flip` *protocol is a correct Byzantine agreement protocol that halts in $t+1$ rounds in the worst case, and is polynomial in both communication and computation.*

9. Early stopping. The `Sliding-flip` protocol consists of an ensemble of agreement processes running concurrently. They are initiated one per round, and are combined using the `Preempt-on-One` scheme: a local decision of 1 in any instance causes a global decision of 1 coupled with preemption of all decision processes. A decision of 0 can be reached only at time $t+1$, in case all agreement processes turn out to have decided 0. (Recall that the agreement processes in the ensemble are all guaranteed to reach a local decision by the end of round $t+1$.) In particular, even in runs with no failures, a decision of 0 cannot take place before round $t+1$. Thus, while `Sliding-flip` is a polynomial protocol that is guaranteed to halt in $t+1$ rounds, it is not guaranteed to stop early in runs in which few processors actually fail. In this section we discuss how to modify the `Sliding-flip` protocol to obtain a protocol that does stop early when few failures actually occur. The concept of *early stopping* is due to [13], who showed that no protocol for Byzantine agreement can be guaranteed to halt in fewer than $\min\{t+1, f+2\}$ rounds in the worst case, where f is the number of failures that occur in the run in question. Our goal will be to obtain a protocol that is guaranteed to halt in $\min\{t+1, f+c\}$ rounds for a small constant c.

The `Preempt-on-One` scheme already takes care of stopping quickly when the decision is 1. Early decision on 1 was of crucial importance to the `Sliding-flip` protocol, because that was the way the protocol keeps trees from growing beyond a polynomial bound. What remains, therefore, is to allow early stopping on a decision

of 0 without hindering the correctness of the early decision on 1. Our basic strategy will be to maintain the basic `Preempt-on-One` rules for deciding on 1. Early stopping on 0 will then depend on our ability to predict at an early stage that all agreement processes that have been initiated, as well as all those that are due to be initiated in the future, are bound to decide 0. It is safe to decide 0 when this happens, rather than wait until the end of round $t+1$ to do so. The following lemma describes a fairly general condition that guarantees that all future monitors will decide 0.

LEMMA 9.1. *In the protocol* `Sliding-flip`*, if all monitors* $\mathtt{M}^R$ *with* $R \leq K$ *are closed on* 0 *for all nonfaulty processors at the end of round* K*, then every monitor* $\mathtt{M}^R$ *with* $R > K$ *that is initiated in the protocol closes on* 0 *at the end of its first round* R*.*

Proof. Assume that all monitors $\mathtt{M}^R$ with $R \leq K$ are closed on 0 for all nonfaulty processors at the end of round K. Part (i) of the monitor-vote rule implies that all nonfaulty processors will vote 0 on the monitor $\mathtt{M}^R$ for the first $R > K$. This monitor will close on 0 in its first round for all nonfaulty processors, and the same argument can now be applied inductively to show that all later monitors will do the same. □

The problem in trying to apply the condition of Lemma 9.1 is that this condition is not one that can be detected by an individual processor. We now discuss a way of making a similar condition detectable. In order to do so, we consider a variant `Sliding-flip`$'$ of the `Sliding-flip` protocol that differs from the original only in that, instead of initiating a new monitor agreement process $\mathtt{M}^R$ in every round, such a process is initiated once in five rounds. Specifically, $\mathtt{M}^R$ monitors are initiated for every integer R of the form $R = 5k+1$, for $1 \leq k \leq \lfloor \frac{t-1}{5} \rfloor$. (The agreement tree based on the processors' original initial values is, of course, initiated in round one.) First notice that `Sliding-flip`$'$ has all of the desired qualities of `Sliding-flip`: It halts in at most $t+1$ rounds, and is guaranteed to be polynomial. The proof of correctness of `Sliding-flip`$'$ is the same as that for `Sliding-flip`. The only changes required in order to prove the complexity bounds for `Sliding-flip`$'$ are in Corollary 7.11, Lemma 8.6, and Lemma 8.7. In Lemma 8.6 we are now guaranteed only that if $\mathtt{G\_Waste}(R) \geq 6$ then a monitor issued no later than round $R+7$ closes with value 1. Without any fine tuning, this will cause an increase of $O(n^4)$ in the complexity of the protocol. The analogue of Corollary 7.11 will replace $r-6$ by $r-10$ and n^6T by $n^{10}T$. The proof of the analogue of Lemma 8.7 goes through essentially without change, when we assume $\mathtt{G\_Waste}(R) \leq 6$ instead of assuming that $\mathtt{G\_Waste}(R) \leq 2$.

Define $\Phi_i(K)$ to hold if (i) $K \equiv 1 \bmod 5$, (ii) no node of (global) depth K is corrupted in any of i's trees, and (iii) no agreement process initiated in a round $R \leq K$ either has or will close with the root fixed to value 1. Intuitively, we will use $\Phi_i(K)$ to determine a condition for early stopping on 0. We can show the following lemma.

LEMMA 9.2. *If* $\Phi_i(K)$ *holds for some* K*, then all monitors* $\mathtt{M}^R$ *for* $R > K$ *that will ever be initiated in* `Sliding-flip`$'$ *will close with value* 0 *in round* R*, for all nonfaulty processors.*

Proof. Let $K \equiv 1 \bmod 5$, and assume that no node of (global) depth K is corrupted in any of i's trees. Notice that because $K \equiv 1 \bmod 5$, no monitors are initiated in the rounds $K+1, \ldots, K+4$. By definition of corruption, the fact that no node of (global) depth K is corrupted for i implies that all depth K nodes in all of i's trees are closed by the end of round $K+2$. Corollary 6.16 implies that all depth K nodes in the trees of all nonfaulty processors are closed by the end of round $K+4$. It follows that all trees corresponding to existing agreement processes close for all nonfaulty processors by that time. As a result, by part (i) of the monitor-vote rule, all

nonfaulty processors will vote 0 on $\mathtt{M}^R$ for $R = K+5$, and a straightforward inductive argument shows that they will all vote 0 on every monitor initiated thereafter. □

The condition $\Phi_i(K)$ used in Lemma 9.2 is easily detectable by process i. Moreover, Lemma 9.2 implies that if Φ_i holds, then Φ_j will hold soon thereafter.

COROLLARY 9.3. *Let i and j be nonfaulty processors. If $\Phi_i(K)$ holds in a run of* `Sliding-flip′`*, then $\Phi_j(K+5)$ holds at the end of round $K+5$.*

Proof. Assume that $\Phi_i(K)$ holds. The argument given in the proof of Lemma 9.2 shows that the trees corresponding to all monitors initiated in rounds $R \leq K$ will be closed for all nonfaulty processors by the end of round $K+4$. By the monitor-vote rule, all nonfaulty processors vote 0 on the monitor initiated in round $K+5$. As a result, this monitor is closed at the end of round $K+5$, for all nonfaulty processors. It follows that $\Phi_j(K+5)$ holds, for all nonfaulty processors j. □

Notice that if $\Phi_i(K)$ holds, then i will detect that $\Phi_i(K)$ holds no later than by the end of round $K+2$. Moreover, the proof of Corollary 9.3 shows that $\Phi_j(K+5)$ will be detectable by j no later than by the end of round $K+5$. It follows that once i detects that $\Phi_i(K)$ holds, it can essentially decide 0. The only thing it needs to do in order to guarantee that Φ_j will hold for all j is to continue to participate in the existing agreement processes (all of which, by Corollary 6.17, are guaranteed to terminate by the end of round $K+4$), and send its vote in the monitor initiated in round $K+5$. This provides an early-stopping method for deciding 0.

We define the `ES-Sliding-flip` protocol to consist of `Sliding-flip′` with the following modifications: (i) we add the rule that once a processor i first detects that $\Phi_i(K)$ holds, it decides 0 and it halts once all of its monitors initiated in rounds $R \leq K$ halt according to Δ-ES; and (ii) we add the default that sending no value as a vote on a monitor amounts to sending a vote of 0 (and is not regarded an ill-formed message). The purpose of point (ii) is to enable a processor that detects that $\Phi_i(K)$ holds to halt before round $K+5$. Its silence in round $K+5$ will be interpreted as a vote of 0, which will suffice to ensure that the necessary conditions of $\Phi_j(K+5)$ will hold. We can now summarize the properties of the protocol `ES-Sliding-flip` as follows.

THEOREM 9.4. *The* `ES-Sliding-flip` *protocol is a correct Byzantine agreement protocol that is polynomial in both communication and computation. It halts in* $\min\{t+1, f+5\}$ *rounds in the worst case, where f is the number of failures that actually take place.*

Proof. Correctness of the protocol `ES-Sliding-flip` is inherited from that of `Sliding-flip′` and Corollary 9.3, which guarantees that if one of the nonfaulty processors decides 0, then all of them will. The polynomial complexity and $(t+1)$-boundedness of `ES-Sliding-flip` are also inherited from `Sliding-flip′`. It remains to show that the protocol halts in no more than $f+5$ rounds. First notice that for a node of global depth K to be corrupted, at least K distinct processors must fail. This is obvious in the initial agreement process, and is true for arbitrary monitors $\mathtt{M}^R$ based on Lemma 6.1 and part (ii) of the monitor-vote rule. Assume that exactly f processors fail in a given run. It follows that no more than f processors can ever be disabled, and no node of depth greater than f can be corrupted. It follows that by the end of round $f+3$ all monitors ever initiated are closed in all nonfaulty processors' trees. Since exactly f processors fail, if a node of depth f is corrupted, no nonfaulty processor can detect f processors as being disabled before round $f+2$, because an additional round in which `mask` messages are sent is necessary for the detection. This implies that the last monitor on which some nonfaulty processor votes 1 can

be initiated in a round $K \leq f$. Let $F = \min\{K : K \equiv 1 \bmod 5 \,\&\, K > f\}$. Notice that $F \leq f + 5$. Let i be an arbitrary nonfaulty processor. If $\Phi_i(K)$ holds for some $K < F$, then we have seen that i detects $\Phi_i(K)$ no later than time $K + 2$, and halts no later than time $K + 4$, and $K + 4 < F \leq f + 5$. Assume that $\Phi_i(K)$ did not hold for $K < F$. It follows that $\Phi_i(F)$ will hold at the end of round $f + 3$. Processor i will be able to decide 0 at the end of round $f + 3$, and halt at the end of round $f + 5$. It follows that all processors decide and halt by the end of round $f + 5$ and we are done. □

A few remarks are in order: (i) It is possible to use the reconstruction method of [26, 28, 7] in order to avoid the need for the additional two rounds in which nonfaulty processors "echo" values to ensure that the others reach the same decision in an instance of Δ-ES. A variant of `ES-Sliding-flip` based on such a modified version of Δ-ES will halt in $\min\{t + 1, f + 3\}$ rounds.

(ii) To obtain an early-stopping protocol halting within $f + 2$ rounds, what is needed is to extend the fixing rule `Fx2` in order to allow fixing in round $|\sigma| + 1$ to the nonfavored value $1 - \mathtt{par}(|\sigma|)$ if an overwhelming majority of σ's children store this value. The current version allows fixing only to $\mathtt{par}(|\sigma|)$. The current choice was made in order to simplify the statements of the various lemmas in this paper, and to shorten their proofs. We believe that with this extension and using reconstruction, it should be possible to obtain an early stopping protocol that halts in the optimal bounds of $\min\{t + 1, f + 2\}$ rounds.

(iii) In runs with no failures, our protocol decides in at most three rounds and halts in five. Only a single agreement tree is ever constructed, and the protocol acts just like a simple straightforward protocol. With reconstruction the protocol would halt in three rounds, and with an extended `Fx2` rule both decision and termination would be obtained in two rounds.

Acknowledgments. The authors would like to thank Café Ka'ze in Tel Aviv for providing the ambiance and hospitality during critical stages of this work. Special thanks to Cynthia Dwork, Arkady Zamsky, and to an anonymous referee for comments and suggestions that improved this paper. The first author is also thankful to Piotr Berman for innumerable discussions on the subject, and the second author is similarly thankful to Orli Waarts.

REFERENCES

[1] A. Bar-Noy and D. Dolev, *Consensus algorithms with one-bit messages*, Distrib. Comput., 4 (1991), pp. 105–110. Preliminary version appeared as *Families of consensus algorithms*, in Proc. 3rd Aegean Workshop on Computing, Lecture Notes in Comput. Sci. 319, Springer-Verlag, New York, 1988, pp. 380–390.

[2] A. Bar-Noy, D. Dolev, C. Dwork, and H. R. Strong, *Shifting gears: Changing algorithms on the fly to expedite Byzantine agreement*, Inform. and Comput., 97 (1992), pp. 205–233.

[3] P. Berman and J. A. Garay, *Cloture votes: $n/4$-resilient distributed consensus in $t + 1$ rounds*, Math. Systems Theory, Ray Strong, ed., 26 (1993), pp. 3–20; special issue dedicated to fault-tolerant distributed algorithms.

[4] P. Berman and J. A. Garay, *Efficient distributed consensus with $n = (3+\epsilon)t$ processors*, in Proc. 5th Internat. Workshop on Distributed Algorithms, Lecture Notes in Comput. Sci. 579, Springer-Verlag, New York, 1991, pp. 129–142.

[5] P. Berman and J. A. Garay, private communication, 1992.

[6] P. Berman, J. A. Garay, and K. J. Perry, *Towards optimal distributed consensus*, in Proc. 30th IEEE Symposium on the Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1989, pp. 410–415.

[7] P. Berman, J. A. Garay, and K. J. Perry, *Optimal early stopping in distributed consensus*, in Proc. 6th Internat. Workshop on Distributed Algorithms, Lecture Notes in Comput. Sci. 647, Springer-Verlag, New York, 1992, pp. 221–237.

[8] B. Coan, *A communication-efficient canonical form for fault-tolerant distributed protocols*, in Proc. 5th ACM Symposium on the Principles of Distributed Computing, ACM, New York, 1986, pp. 63–72.

[9] B. Coan, *Efficient agreement using fault diagnosis*, Distrib. Comput., 7 (1993), pp. 87–98; also in Proc. 26th Allerton Conference on Communication, Control and Computing, University of Illinois, Urbana, IL, 1988, pp. 663–672.

[10] B. Coan and J. Welch, *Modular construction of an efficient* 1*-Bit Byzantine agreement protocol*, Math. Systems Theory, H. R. Strong, ed., 26 (1993), pp. 131–154; special issue dedicated to fault-tolerant distributed algorithms.

[11] D. Dolev, M. J. Fischer, R. Fowler, N. A. Lynch, and H. R. Strong, *An efficient algorithm for Byzantine agreement without authentication*, Inform. and Control, 52 (1982), pp. 257–274.

[12] C. Dwork and Y. Moses, *Knowledge and common knowledge in a Byzantine environment: Crash failures*, Inform. and Comput., 88 (1990), pp. 156–186.

[13] D. Dolev, R. Reischuk, and H. R. Strong, *Early Stopping in Byzantine Agreement*, IBM Research Report RJ5406 (55357), IBM Almaden Research Center, San Jose, CA, 1986; revised version appears in J. Assoc. Comput. Mach., 37 (1990), pp. 720–741.

[14] D. Dolev and H. R. Strong, *Polynomial algorithms for multiple processor agreement*, in Proc. 14th Annual Symposium on Theory of Computing, ACM, New York, 1982, pp. 401–407.

[15] M. J. Fischer, *The Consensus Problem in Unreliable Distributed Systems (A Brief Survey)*, Yale University Technical Report YALEU/DCS/RR-273, New Haven, CT, 1983.

[16] M. J. Fischer and N. A. Lynch, *A lower bound for the time to assure interactive consistency*, Inform. Process. Lett., 14 (1982), pp. 183–186.

[17] P. Feldman and S. Micali, *Optimal algorithms for Byzantine agreement*, in Proc. of the 20th Annual Symposium on Theory of Computing, ACM, New York, 1988, pp. 148–161.

[18] J. A. Garay and Y. Moses, *Fully polynomial Byzantine agreement in t+1 rounds*, in Proc. of the 25th Annual ACM Symposium on Theory of Computing, ACM, New York, 1993, pp. 31–41.

[19] O. Goldreich and E. Petrank, *The best of both worlds: Guaranteeing termination in fast randomized Byzantine agreement protocols*, Inform. Process. Lett., 36 (1990), pp. 45–49.

[20] L. Lamport, R. E. Shostak, and M. Pease, *The Byzantine generals problem*, ACM Trans. Prog. Lang. and Systems, 4 (1982), pp. 382–401.

[21] Y. Moses and O. Waarts, *Coordinated traversal: $(t+1)$-round Byzantine agreement in polynomial time*, J. Algorithms, 17 (1994), pp. 110–156; an extended abstract appeared in Proc. 29th IEEE Symposium on the Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 246–255.

[22] M. Pease, R. Shostak, and L. Lamport, *Reaching agreement in the presence of faults*, J. Comput. Mach., 27 (1980), pp. 121–169.

[23] M. Rabin, *Randomized Byzantine generals*, in Proc. 24th IEEE Symposium on the Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1983, pp. 403–409.

[24] S. Toueg, K. J. Perry, and T. K. Srikanth, *Fast distributed agreement*, SIAM J. Comput., 16 (1987), pp. 445–457.

[25] O. Waarts, *Coordinated traversal: Byzantine Agreement in Polynomial Time*, M. Sc. thesis, Weizmann Institute of Science, Rehovot, Israel, 1988.

[26] A. Zamsky, *New Algorithms for Agreement in Synchronous Distributed Networks*, M. Sc. thesis, Technion—Israel Institute of Technology, Haifa, Israel, 1992.

[27] A. Zamsky, *A randomized Byzantine agreement protocol with constant expected time and guaranteed termination in optimal (deterministic) time*, in Proc. 15th Annual ACM Symposium on Priniciples of Distributed Computing, ACM, New York, 1996, pp. 201–208.

[28] A. Zamsky, A. Israeli, and S. Pinter, *Optimal time Byzantine agreement for $t < n/8$*, Distrib. Comput., 9 (1995), pp. 95–108.

[29] H. Attiya, C. Dwork, N. Lynch, and L. Stockmeyer, *Bounds on the time to reach agreement in the presence of timing uncertainty*, J. Assoc. Comput. Mach., 41 (1994), pp. 122–152.

SIAM J. COMPUT.
Vol. 27, No. 1, pp. 291–301, February 1998
© 1998 Society for Industrial and Applied Mathematics
012

AN $O(\log k)$ APPROXIMATE MIN-CUT MAX-FLOW THEOREM AND APPROXIMATION ALGORITHM*

YONATAN AUMANN† AND YUVAL RABANI‡

Abstract. It is shown that the minimum cut ratio is within a factor of $O(\log k)$ of the maximum concurrent flow for k-commodity flow instances with arbitrary capacities and demands. This improves upon the previously best-known bound of $O(\log^2 k)$ and is existentially tight, up to a constant factor. An algorithm for finding a cut with ratio within a factor of $O(\log k)$ of the maximum concurrent flow, and thus of the optimal min-cut ratio, is presented.

Key words. approximation algorithms, cuts, sparse cuts, network flow, multicommodity flow

AMS subject classifications. 05C38, 68R10, 90B10

PII. S0097539794285983

1. Introduction.

1.1. Multicommodity flow. Consider an undirected graph $G = (V, E)$ with an assignment of nonnegative capacities to the edges, $c : E \to \mathbb{R}^+$. A *multicommodity flow* instance on G is a set of ordered pairs of vertices $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$. Each pair (s_i, t_i) represents a *commodity*, with *source* at s_i and *destination* or *target* at t_i. The s_i's and t_i's are also called *terminals*. The objective is to maximize the amount of *flow* traveling from the sources to the corresponding destinations, subject to the capacity constraints. The problem comes in two flavors. In the first, called the *maximum throughput* problem, the total flow, summed over all commodities, is to be maximized. The second is called the *maximum concurrent flow* problem. Here, for each commodity (s_i, t_i) a nonnegative demand D_i is specified. The objective is to maximize the *fraction* of the demand that can be shipped simultaneously for all commodities. A maximum concurrent flow instance is *uniform* if the set of commodities is the set of all ordered pairs of vertices and all demands are equal. Both the maximum throughput problem and the maximum concurrent flow problem can be solved in polynomial time using linear programming.

Given a multicommodity flow instance (together with demands) the *minimum cut ratio* is defined. A *cut* $(S, \overline{S})$ is a partition of the vertices, with $S \cup \overline{S} = V$ and $S \cap \overline{S} = \emptyset$. The *capacity* of the cut $(S, \overline{S})$ is the sum of capacities of the edges with one endpoint in S and the other in $\overline{S}$. The *cut ratio* is this capacity divided by the sum of demands of commodities with one terminal in S and the other terminal in $\overline{S}$. Finally, the minimum cut ratio R is the minimum of cut ratios taken over all cuts $(S, \overline{S})$,

$$R = \min_{S \subset V} \frac{\sum_{e \in E \cap (S \times \overline{S})} c(e)}{\sum_{(s_i, t_i) \in (S \times \overline{S}) \cup (\overline{S} \times S)} D_i} .$$

*Received by the editors October 7, 1994; accepted for publication (in revised form) January 19, 1996.

http://www.siam.org/journals/sicomp/27-1/28598.html

†Department of Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel (aumann@cs.biu.ac.il).The work of this author was done while visiting MIT Laboratory for Computer Science, and was supported by the Wolfson postdoctoral fellowship and DARPA contract N00014-92-J-1799.

‡Computer Science Department, The Technion, Haifa 32000, Israel (rabani@cs.technion.ac.il).The work of this author was done while visiting MIT Laboratory for Computer Science, and was supported by ARPA/Army contract DABT63-93-C-0038.

In general, determining the minimum cut ratio is an NP-hard problem [9]. The maximum concurrent flow is a lower bound on the minimum cut ratio.

For the maximum throughput problem a different notion is useful—that of the *minimum multicut*. A *multicut* is a subset of edges $F \subseteq E$ whose removal disconnects all source-destination pairs. The *capacity* of a multicut F is the sum of capacities of the edges in F. Determining the value of the minimum multicut is an NP-hard and a MAX-SNP-hard problem [11]. The maximum throughput is a lower bound on the capacity of the minimum multicut.

Our results. In this paper, we study the problem of determining the worst-case ratio between the minimum cut ratio and the maximum concurrent flow. We establish that for any k commodity instance, the ratio is $O(\log k)$. The bound holds for instances involving arbitrary capacities and demands, thus improving upon the best previously known bound of $O(\log^2 k)$ [27]. We also consider the related question of finding a cut whose ratio approximates the minimum cut ratio. Finding such a cut is a basic step in approximation algorithms for many NP-hard problems (see [20, 16]). We give an algorithm that produces a cut whose ratio is within an $O(\log k)$ factor of the maximum concurrent flow, and thus within at most this factor of the minimum cut ratio.

We now give a brief overview of the proof method and the procedure for obtaining the cut. Starting with the linear programming formulation of the maximum concurrent flow problem, a solution to the dual program is an assignment of points in $\mathbb{R}^k$ to the vertices of the graph. The *value* of the solution is the sum of L_∞ distances spanned by the edges, scaled by the capacities. The constraints of the dual program dictate that the sum of distances between source-destination pairs, scaled by the demands, equals one. Thus, we first solve the (dual) linear program and obtain an embedding of the graph in $\mathbb{R}^k$. Next, we wish to produce the cut by placing a hyperplane in the space, where the two sides of the hyperplane determine the cut. Suppose that a random hyperplane would have the property that for each pair of vertices x, y, the hyperplane separates x from y with probability proportional to the distance between the two. Then, the expected amount of demands separated by a random hyperplane would be 1 (the sum of source-destination distances scaled by the demands). The expected capacity of the cut, induced by the random hyperplane, would be the sum of edge distances scaled by the capacities, which is exactly the *value* of the solution to the dual, which, in turn, equals the optimal value of the primal, which is the max-flow. Given these expectation, we could then try and find a hyperplane which maintains the expected ratio. Such a cut would have cut ratio equal to the max-flow. Clearly, this procedure fails. This is because in L_∞ norm the necessary distribution over hyperplanes does not necessarily exist. As it turns out, L_1 is the natural norm under which to produce such a distribution. In particular, in L_1 norm a random hyperplane perpendicular to one of the axes has exactly the necessary property (i.e., it separates any two nodes with probability proportional to the distance between the two).

With this intuition in mind, we use small distortion embeddings of finite metric spaces into ℓ_1 [3, 21]. Once the (solution) graph is embedded in ℓ_1, we show how to find (deterministically) a good cut. The factor of $O(\log k)$ is lost in the distortion of the embedding. We also show that similar ideas are applicable to bounding the min-multicut max-flow ratio for the maximum throughput problem, though the bounds derived are inferior to the best-known bounds. Finally, as a consequence of known instances of graphs with a high min-cut max-flow ratio, we derive tight (up to a constant factor) lower bounds on the best possible distortion of embeddings into ℓ_1

and into ℓ_2, thus improving the lower bound of [3].

Related work. One of the most celebrated theorems in combinatorial optimization is the min-cut max-flow theorem of Ford and Fulkerson and of Elias, Feinstein, and Shannon [8, 7], which states that for a single commodity, the maximum flow is equal to the minimum cut (here the two flavors of the problem coincide). Early on it had been noted that this is not the case for multicommodity flow.

Early work on multicommodity flow concentrated on characterizing instances where the maximum flow equals the minimum cut ratio. Equality has been established for instances where the *support* (i.e., the graph created by connecting source-destination pairs) is either the union of two stars, or the clique K_4, or the cycle C_5, through the consecutive works of Hu [14], Rothschild and Whinston [28], Dinits (see [1]), Seymour [29], and Lomonosov [23]. Seymour [30] shows that equality holds when the union of the network and the support is planar. Okamura and Seymour [26] establish equality for instances on planar graphs where all terminals reside on the boundary of a single face. For more on this vein of work see [18].

In their groundbreaking work, Leighton and Rao [20] introduce the notion of an approximate min-cut max-flow theorem. They show that for uniform multicommodity flow, the min-cut ratio is within a factor of $O(\log n)$ of the maximum concurrent flow, where n is the number of nodes in the network. Klein et al. [16] extend their techniques to show an $O(\log C \log D)$ approximate min-cut max-flow theorem for general concurrent flow, where C is the sum of capacities and D is the sum of demands. This ratio has been improved to $O(\log^2 k)$, where k is the number of commodities, through the works of Tragoudas [32], Garg, Vazirani, and Yannakakis [12], and Plotkin and Tardos [27]. The prevalent method in these papers is the use of graph partitions (see [17]). The last paper in the sequence overcomes the dependency on the values of the demands by applying a scaling argument and combining flows. We note that while in [20, 16] the primal program considered is the problem of minimizing the capacity utilization (i.e., the factor by which edge capacities need to be multiplied to allow all the demand through; see also Shahrokhi and Matula [31]), in [12] the primal program considered is that of maximizing the throughput. This latter view is the one taken in this paper.

Multicuts have been considered by Garg, Vazirani, and Yannakakis [11, 12]. The first paper gives a 2-approximate min-multicut max-flow bound and approximation algorithm for maximum throughput instances on arbitrary capacitated trees. The second paper gives an $O(\log k)$-approximate min-multicut max-flow bound and approximation algorithm for arbitrary maximum throughput instances. Dahlhaus et al. [5] consider the problem of *multiway cuts*, i.e., multicuts for maximum throughput instances defined by listing all pairs among a subset of k vertices. They show a $2 - \frac{2}{k}$ approximation algorithm.

Of related interest is the beautiful approximate max-cut algorithm given by Goemans and Williamson [13]. They use semidefinite programming to optimize distances among vertices of a graph. When embedded into the unit ball in $\mathbb{R}^n$ with distances determined by the L_2 norm, the original distances do not shrink too much (but they may have an unbounded increase). It is easy to cut the embedded graph by a hyperplane. Similar ideas have been exploited by Karger, Motwani, and Sudan to give an improved approximation algorithm for vertex coloring [15]. We note that these techniques seem to fail when it is required to minimize a cut rather than to maximize it.

The relation between ℓ_1-embeddability and multicommodity flow has been noted

by Avis and Deza [2]. They show that Lomonosov's min-cut-equals-max-flow result (and its predecessors for two commodities, etc.) is related directly to the ℓ_1-embeddability of certain fixed metric spaces.

A central tool in our proof is the small distortion embedding of a given metric space into ℓ_1. For this we use a result of Linial, London, and Rabinovich. In a fundamental work, they study algorithmic and other applications of embeddings of graphs into low-dimensional normed spaces, introducing techniques from functional analysis. In particular, they consider a theorem of Bourgain [3] which asserts that any n-point metric space can be embedded into ℓ_1 (and also into ℓ_2) with logarithmic distortion. Bourgain's original proof is existential. Linial, London, and Rabinovich present an algorithmic version of the proof which also bounds the dimension by $O(\log^2 n)$ (see also Matoušek [25]). Following their initial work, Linial, London, and Rabinovich have obtained independently similar results to the ones reported here. A full account of their work, which also includes the earlier results, appears in [21]. We note that, while the derivation in [21] is very similar to ours, we generate the cut by cutting with a hyperplane, whereas in [21] the cut is generated by a different procedure.

Subsequent to this work, Linial, London, and Rabinovich (in the journal version of their paper [22]), and independently Garg [10], have come up with modified algorithms that can be effectively derandomized.

2. Min-cut-ratio max-concurrent-flow. Let $G = (V, E)$ be a graph. Let $c : E \to \mathbb{R}^+$ be an assignment of nonnegative capacities to the edges of G. Consider a maximum concurrent flow problem on G, with demand D_i from source s_i to destination t_i, $i = 1, \ldots, k$. For each i, let $\{q_1^i, q_2^i, \ldots\}$ be an enumeration of the paths from s_i to t_i. Let $q_j^i(e)$ denote the characteristic function of the predicate $e \in q_j^i$. Let f_j^i be the amount of commodity i flowing along the path q_j^i, and let f be the minimum fraction of any of the demands which is transmitted in the system in total. We formulate the maximum concurrent flow problem as the following linear program:

$$\text{(P1)} \qquad \begin{array}{lll} \text{maximize} & f & \text{subject to} \\ & D_i f - \sum_j f_j^i \le 0 & \forall i \in \{1, 2, \ldots, k\}, \\ & \sum_{i,j} q_j^i(e) f_j^i \le c(e) & \forall e \in E, \\ & f_j^i \ge 0 & \forall i, j. \end{array}$$

The first set of constraints requires that the total flow of commodity i be at least an f fraction of the demand D_i. The second set of constraints prevents the edge capacities from being violated.

The dual is

$$\text{(D1)} \qquad \begin{array}{lll} \text{minimize} & \sum_e c(e) d(e) & \text{subject to} \\ & \sum_i D_i h_i = 1, & \\ & \sum_e q_j^i(e) d(e) - h_i \ge 0 & \forall i, j, \\ & h_i \ge 0 & \forall i, \\ & d(e) \ge 0 & \forall e. \end{array}$$

The dual has a pictorial physical interpretation. Imagine that the edges of the graph represent a network of pipes. The "cross-section" area of pipe e is $c(e)$, and $d(e)$ denotes the length of the pipe. Thus, the objective is to minimize the total volume of the network. The variable h_i corresponds to the distance between the terminals of commodity i (i.e., the minimum length of a path q_j^i). Now suppose that the network supports a flow such that for each commodity i, the rate at which this

commodity is transmitted from source to destination is an f fraction of the demand of the commodity. Then, at any given time the total *volume* of flow in the network is at least $\sum_i fD_ih_i$. Clearly, the volume of flow cannot be greater than the total volume of the pipes, which is $\sum_e c(e)d(e)$. The first constraint in the dual scales $\sum_i D_ih_i = 1$. Thus we have $f \leq \sum_e c(e)d(e)$, which is precisely the statement of weak duality in this case.

The program (D1) may be of exponential size. Using the above interpretation, we convert it to an equivalent program of polynomial size. The purpose of this step is to allow the solution of the linear program in polynomial time. We consider the vertices of G as points in $\mathbb{R}^k$ with L_∞ norm. The distances between the vertices are the distances assigned to the corresponding edges. For each $i = 1, \ldots, k$, the ith coordinate of the location of the vertex represents the distances of the vertex from the source s_i. Thus, we obtain the following equivalent, polynomial size, linear program:

$$
\begin{array}{lllll}
\text{minimize} & \sum_e c(e)d(e) & & \text{subject to} & \\
(\mathrm{D}') & \begin{array}{rcl} d(e) & \geq & u_i^x - u_i^y \\ & \geq & u_i^y - u_i^x \end{array} & & \forall e = (x,y),\ \forall i \in \{1,2,\ldots,k\}, \\
& \sum_i D_ih_i = 1, & & & \\
& h_i \leq u_i^{t_i} - u_i^{s_i} & & \forall i, & \\
& u_i^{t_i} \geq 0,\ u_i^{s_i} \leq 0 & & \forall i. &
\end{array}
$$

Here, vertex x is mapped to the point $u^x = (u_1^x, \ldots, u_k^x) \in \mathbb{R}^k$.

LEMMA 2.1. *The optimal solutions to* D1 *and* D′ *are equal.*

Proof. Consider a feasible solution to D′. The same $d(e)$'s and h_i's also constitute a feasible solution for D1. To see this we only have to show that $\sum_e q_j^i(e)d(e) \geq h_i$ for all i, j. Indeed,

$$\sum_e q_j^i(e)d(e) \geq \sum_{e=(x,y)} q_j^i(e)\,|u_i^x - u_i^y| = u_i^{t_i} - u_i^{s_i} \geq h_i\ .$$

The equality holds because q_j^i is a path from s_i to t_i and thus the summation is telescopic. Thus, any feasible solution to D′ is also a solution to D1. Conversely, given a feasible solution to D1, for all vertices x and $i = 1, \ldots, k$, set $u_i^x = \mathrm{dist}(s_i, x)$, where $\mathrm{dist}(y, x)$ denotes the length of the shortest path from y to x, under the non-negative edge lengths $d(e)$. This assignment gives D′ the same objective function value as for D1 while obeying all the constraints. ∎

A feasible solution to D' induces a mapping of the vertices into the metric space ℓ_∞^k ($\mathbb{R}^k$ with L_∞ norm). The L_∞ distances among these points have the following properties:

1. $\forall i$, $\|u^{t_i} - u^{s_i}\|_\infty \geq h_i$; and
2. $\forall e = (x,y)$, $\|u^x - u^y\|_\infty \leq d(e)$.

Next we embed this n point subspace of ℓ_∞^k into ℓ_1^d ($\mathbb{R}^d$ with L_1 norm), with u^x mapped to $\tilde{u}^x$. Let $\tilde{h}_i$ denote $\|\tilde{u}^{s_i} - \tilde{u}^{t_i}\|_1$. The embedding has the following properties:

1. d is polynomial in the size of the input (in fact, the embedding will have $d = O(\log^2 k)$);
2. $\forall i$, $\tilde{h}_i \geq h_i / O(\log k)$;
3. $\forall e = (x,y)$, $\|\tilde{u}^x - \tilde{u}^y\|_1 \leq \|u^x - u^y\|_\infty$.

Such an embedding can be obtained by a slight variation of Bourgain's theorem [3], and the algorithmic version thereof by Linial, London, and Rabinovich. A sketch of the proof is provided in the appendix (Corollary A.2; see also [21]).

For every pair of vertices x, y define the function $\delta^{x,y} : \{1, \ldots, d\} \times \mathbb{R} \to \{0, 1\}$ as

$$\delta^{x,y}(j, \xi) = \begin{cases} 1 & \text{if } \min\{\tilde{u}^x_j, \tilde{u}^y_j\} \le \xi < \max\{\tilde{u}^x_j, \tilde{u}^y_j\}, \\ 0 & \text{otherwise.} \end{cases}$$

The function $\delta^{x,y}$ is the characteristic function of the projection, on the jth axis, of the straight line connecting $\tilde{u}^x$ and $\tilde{u}^y$.

Now, define two functions

$$H : \{1, 2, \ldots, d\} \times \mathbb{R} \to \mathbb{R}^+,$$
$$C : \{1, 2, \ldots, d\} \times \mathbb{R} \to \mathbb{R}^+$$

as follows:

$$H(j, \xi) = \sum_{i=1}^{k} D_i \delta^{s_i,t_i}(j, \xi),$$
$$C(j, \xi) = \sum_{e=(x,y)\in E} c(e)\delta^{x,y}(j, \xi).$$

Notice that $H(j, \xi)$ is the sum of demands cut by a hyperplane perpendicular to the jth axis at ξ, while $C(j, \xi)$ is the sum of capacities of edges cut by the same hyperplane.

We have that

$$\sum_{j=1}^{d} \int_{-\infty}^{\infty} H(j, \xi) d\xi = \sum_{j=1}^{d} \sum_{i=1}^{k} \int_{-\infty}^{\infty} D_i \delta^{s_i,t_i}(j, \xi) d\xi = \sum_{i=1}^{k} D_i \tilde{h}_i \ge \frac{1}{O(\log k)} \,. \tag{2.1}$$

Similarly,

$$\sum_{j=1}^{d} \int_{-\infty}^{\infty} C(j, \xi) d\xi \le \sum_{e\in E} c(e)d(e). \tag{2.2}$$

Therefore, since both functions are nonnegative, there exists a point (j_0, ξ_0) such that $H(j_0, \xi_0) > 0$ and

$$\frac{C(j_0, \xi_0)}{H(j_0, \xi_0)} \le O(\log k) \sum_{e\in E} c(e)d(e).$$

To see this, consider the functions

$$H'(j, \xi) = O(\log k)H(j, \xi)$$

and

$$C'(j, \xi) = \frac{C(j, \xi)}{\sum_{e\in E} c(e)d(e)}.$$

H' is not always 0 and C' is nonnegative. If $H'(j, \xi) < C'(j, \xi)$ whenever $H'(j, \xi) > 0$, then we must have that $\sum_{j=1}^{d} \int_{-\infty}^{\infty} H'(j, \xi) < \sum_{j=1}^{d} \int_{-\infty}^{\infty} C'(j, \xi)$, in contradiction to inequalities (2.1) and (2.2).

The point (j_0, ξ_0) determines a cut $(S, \overline{S})$ by setting $S = \{x \in V \mid \tilde{u}^x_{j_0} \leq \xi_0\}$. The amount of demand between S and $\overline{S}$ is exactly $H(j_0, \xi_0)$ and the capacity of the edges connecting the two sides of the cut is $C(j_0, \xi_0)$. The point (j_0, ξ_0) can be found in polynomial time. This is because there are at most $|V|$ points of interest to check in each dimension. A sweep in each dimension can be used to obtain the best cut.

We have obtained the following theorem.

THEOREM 2.2. *For every concurrent multicommodity flow instance involving k commodities, the minimum cut ratio is within a factor of $O(\log k)$ of the maximum concurrent flow. Finding such a cut can be done in random polynomial time.*

Proof. Obtain the optimal solution to D′, e.g., using Ye's interior point polynomial time algorithm [34]. (Alternatively, find a near-optimal solution using more efficient algorithms, e.g., see Leighton et al. [19].) By Lemma 2.1 and linear programming duality, the value m of this solution is equal to the optimal value of $P1$. The above discussion shows how to find a cut with ratio within $O(\log k)$ of m. □

3. Min-multicut max-throughput. We can apply our method to derive bounds on the min-multicut to maximum-throughput ratio. However, the bounds we derive are inferior to the $O(\log k)$ bound presented in [12]. We get an $O(\log^2 k)$ bound using a randomized algorithm to find an approximate cut. The algorithm in [12] is deterministic. A sketch of the proof follows.

Using the notation of the previous section, we formulate the maximum throughput problem as the following linear program:

$$\text{(P2)} \qquad \begin{array}{lll} \text{maximize} & \sum_{i,j} f^i_j & \text{subject to} \\ & \sum_{i,j} q^i_j(e) f^i_j \leq c(e) & \forall e \in E, \\ & f^i_j \geq 0 & \forall i, j. \end{array}$$

The interpretation of this program is similar to that of (P1). The variables f^i_j denote the flow due to commodity i along the path q^i_j. The constraints guarantee that none of the edge capacities are violated.

The dual is

$$\text{(D2)} \qquad \begin{array}{lll} \text{minimize} & \sum_e c(e) d(e) & \text{subject to} \\ & \sum_e q^i_j(e) d(e) \geq 1 & \forall i, j, \\ & d(e) \geq 0 & \forall e. \end{array}$$

This program can be interpreted similarly to D1. We have a similar network of pipes. The constraints require that the distance between the pair of terminals of any commodity is at least 1. To accommodate a flow of rate $\sum_j f^i_j$ for every commodity i, the total volume of the pipes must be at least $\sum_{i,j} f^i_j$. This demonstrates weak duality.

As above, the dual (D2) can be converted into a polynomial size program where vertices are mapped to points in ℓ^k_∞. Note that in a feasible solution to D2 the distance between *each* source-destination pair is at least 1. We embed the metric space induced by the optimal solution into $\ell_1^{O(\log^2 k)}$, shrinking all distances and maintaining a distance of at least $1/O(\log k)$ among the terminal pairs. For the optimal solution, the embedded graph is contained in a cube of unit side length. Throughout the rest of this section we use the same notation as in the previous section.

Consider the following process. For each dimension j, independently choose a point $\xi_j \in [0, 1]$, uniformly at random. Place a hyperplane intersecting the jth axis at ξ_j and perpendicular to this axis. Consider a pair of vertices x, y. Let $a^{x,y}_j$ denote

the length of the projection, onto the jth axis, of the straight line connecting $\tilde{u}^x$ and $\tilde{u}^y$. Let $p^{x,y}$ be the probability that $\tilde{u}^x$ and $\tilde{u}^y$ are separated by at least one of the $d = O(\log^2 k)$ random hyperplanes. We have that

$$p^{x,y} \le \sum_{j=1}^{d} a_j^{x,y} = \|\tilde{u}^x - \tilde{u}^y\|_1 \le \|u^x - u^y\|_\infty ,$$

and

$$p^{x,y} \ge 1 - \prod_{j=1}^{d} \left(1 - a_j^{x,y}\right) \ge 1 - \left(1 - \frac{\|\tilde{u}^x - \tilde{u}^y\|_1}{d}\right)^d .$$

For s_i, t_i, we have $\|u^{s_i} - u^{t_i}\|_1 \ge 1/O(\log k)$ for all i. Thus, if we repeat the placement of d random hyperplanes $c \log^2 k = O(\log^2 k)$ times, then the probability, p_i, that $\tilde{u}^{s_i}$ and $\tilde{u}^{t_i}$ are separated by any of these hyperplanes satisfies

$$p_i \ge 1 - \left(1 - \frac{1}{d \cdot O(\log k)}\right)^{cd \cdot O(\log^2 k)} \ge 1 - k^{-c \cdot O(1)}.$$

Thus, with c sufficiently large, with probability at least $1 - k^{-1} \ge 2/3$ (assuming $k \ge 3$, for $k = 1$ or $k = 2$ the min-multicut equals the max-throughput), all source-destination pairs are separated.

For any edge $e = (x, y)$, $\|u^x - u^y\|_\infty \le d(e)$. Thus, the probability, p_e, that e is cut by any of the hyperplanes satisfies $p_e \le O(\log^2 k) d(e)$. Therefore, the expected size of the multicut is $O(\log^2 k) \sum_e c(e)d(e)$. By Markov's inequality, the probability that we get a multicut whose size is more than twice the expectation is at most $1/2$. Therefore, with probability at least $1/6$ all commodities are cut and the size of the cut is no more than twice the expectation. This guarantees that in random polynomial time we get a multicut that separates all source-destination pairs and whose size is $O(\log^2 k) \sum_e c(e)d(e)$.

Appendix A. Embeddings of finite metric spaces. An *embedding* of a finite metric space $\mathcal{M} = (X, d)$ into a (larger) metric space $\mathcal{M}' = (X', d')$ is a 1–1 mapping $\varphi : X \to X'$. An embedding φ is a *contraction* if $\forall x, y \in X$, $d'(\varphi(x), \varphi(y)) \le d(x, y)$ (see [33]). Any embedding into a normed space can be converted into a contraction without changing the ratios of distances among the points, by scaling all distances. For a contraction φ, the *distortion* is $\max_{x,y \in X} \{d(x, y)/d'(\varphi(x), \varphi(y))\}$. Obviously, the distortion is at least 1. A contraction is *isometric* if it has distortion exactly 1. For any p, a finite metric space is ℓ_p*-embeddable* if it can be isometrically embedded into the r-dimensional normed space ℓ_p^r, for some r. A finite metric space is *Euclidean* if it is ℓ_2-embeddable. We say that φ is into ℓ_p, if $\mathcal{M}'$ is ℓ_p^r, for some r. For $x \in X$, $Y \subset X$, denote by $d(x, Y)$ the distance from x to Y, defined as $\min_{y \in Y} d(x, y)$. For $x \in X$, $\rho \ge 0$, denote $B(x, \rho) = \{z \mid d(x, z) \le \rho\}$ (the closed ball around x of radius ρ).

The following lemma is due to Linial, London, and Rabinovich [21, 22] (see also [25]). It gives an algorithmic version of a theorem of Bourgain [3]. For completeness, we give a sketch of the proof.

LEMMA A.1 (see [22]). *Let $\mathcal{M} = (X, d)$ be a finite metric space with $|X| = n$. There exists a contraction, φ, of $\mathcal{M}$ into $\ell_1^{O(\log^2 n)}$ that has distortion $O(\log n)$. The contraction can be constructed in random polynomial time.*

Proof. For every $t = 1, \ldots, \log n - 1$, choose $L = O(\log n)$ subsets $Q_{t,j} \subset X$, $j = 1, \ldots, L$, such that $|Q_{t,j}| = \frac{n}{2^t}$. Each subset is chosen uniformly at random among all subsets of the specified size. The choices are mutually independent. For each $x \in X$, define a vector $\phi(x)$ with entries $\phi_{t,j}(x) = d(x, Q_{t,j})$.

Consider two points $x, y \in X$. For $t = 0, 1, \ldots$ define $\rho_t(x) = \min \rho$ such that $|B(x, \rho)| \geq 2^t$, and similarly define $\rho_t(y)$. Let $\rho_t = \max\{\rho_t(x), \rho_t(y)\}$, and let $\hat{t}$ be the largest t such that $\rho_t \leq d(x, y)/3$. Set $\rho_{\hat{t}+1} = d(x, y)/3$. (Notice that $\rho_0 = 0$.) Consider a particular $t > 0$ and one of the corresponding sets $Q_{t,j}$. W.l.o.g., $\rho_t(x) = \rho_t$ (i.e., ρ_t is obtained around x). Then, by assumption $|B(x, \rho_t)| \leq 2^t$, and by definition $|B(y, \rho_{t-1})| \geq 2^{t-1}$. Thus, a random set $Q_{t,j}$ of size $n/2^t$ has a constant probability of intersecting $B(y, \rho_{t-1})$ and not intersecting $B(x, \rho_t)$. (Notice that these balls are disjoint.) Thus, with constant probability $|\phi_{t,j}(x) - \phi_{t,j}(y)| \geq \rho_t - \rho_{t-1}$. Thus, with L sufficiently large, with high probability, $\sum_{j=1}^{L} |\phi_{t,j}(x) - \phi_{t,j}(y)| \geq cL(\rho_t - \rho_{t-1})$ for some constant c. Thus,

$$\|\phi(x) - \phi(y)\|_1 = \sum_{t,j} |\phi_{t,j}(x) - \phi_{t,j}(y)| \geq \sum_{t=1}^{\hat{t}+1} cL(\rho_t - \rho_{t-1}) = cL\rho_{\hat{t}+1} = cL\frac{d(x, y)}{3}.$$

On the other hand, by the triangle inequality, for any $x, y \in X$, and any set $Q_{t,j}$, $|d(x, Q_{t,j}) - d(y, Q_{t,j})| \leq d(x, y)$. Thus,

$$\|\phi(x) - \phi(y)\|_1 = \sum_{t,j} |\phi_{t,j}(x) - \phi_{t,j}(y)| \leq \log n \cdot L \cdot d(x, y) \ .$$

Finally, define $\varphi(x) = \frac{1}{L \log n}\phi(x)$. ∎

We get the following corollary.

COROLLARY A.2. *Let $\mathcal{M} = (X, d)$ be a finite metric space. Let $\mathcal{N} = (Y, d)$ be a subspace of $\mathcal{M}$ induced by a subset of points of cardinality k. There exists a contraction φ of $\mathcal{M}$ into $\ell_1^{O(\log^2 k)}$ such that the restriction of φ to $\mathcal{N}$ has distortion $O(\log k)$. The contraction can be constructed in random polynomial time.*

Proof. Using the construction for $\mathcal{N}$ given by the proof of Lemma A.1, we get the subsets $Q_{t,j} \subset Y$, $t = 1, \ldots, \log k - 1$, $j = 1, \ldots, L$, $L = O(\log k)$. For each $x \in X$, set $\phi_{t,j}(x) = d(x, Q_{t,j})$, and define $\varphi(x) = \frac{1}{L \log k}\phi(x)$. Now, for any $x, y \in X$, $\|\varphi(x) - \varphi(y)\|_1 \leq d(x, y)$, and for all $x, y \in Y$, $\|\varphi(x) - \varphi(y)\|_1 \geq d(x, y)/c \log k$, for some constant c. ∎

THEOREM A.3. *For any n, there exists an (explicitly constructible) metric space $\mathcal{M}_n$ over n points, such that any contraction of $\mathcal{M}_n$ into ℓ_1 has distortion $\Omega(\log n)$.*

Proof. The proof is by contradiction. Consider an n node bounded degree expander graphs G_n. (The expansion property we need is that any subset of $s \leq n/2$ vertices is connected to at least αs other vertices, for some absolute constant $\alpha > 0$. Such graphs can be constructed, e.g., via [24].) There are $\Theta(n)$ edges in G_n, and there exist $k_n = \theta(n^2)$ pairs of vertices at distance $\Theta(\log n)$ apart. For each n, consider a multicommodity flow instance where all the edges of G_n have unit capacity and there is a unit demand between each of the k_n pairs. It is not difficult to see that any cut in G_n has cut ratio at least c/n, for some constant c. Consider a feasible solution to the dual problem, where $d(e)$ is set to $d_n = \Theta(1/k_n \log n)$ for all e with an appropriate choice of constant. The value of the dual for this solution is at most $c'/n \log n$, for some constant c'. Now, consider the path metric of G_n (i.e., the n-point metric space over vertices of G_n, where the distance between two vertices is the minimum length

path connecting them). Suppose this metric space can be embedded into ℓ_1 with distortion less than $c \log n/c'$. Using the arguments from section 2, this implies that there exists a cut in G_n with cut ratio less than c/n, which is a contradiction. □

Remark. An alternative proof can be derived by noticing that for multicuts one can disconnect a constant fraction of the commodities by a multicut whose size is within an order of the distortion factor of the optimal fractional solution. The result then follows since there are instances of multicommodity flow on expander graphs, where cutting a constant fraction of the commodities requires a cut whose size is $\Omega(\log k)$ times the value of the maximum throughput (see [12]).

Bourgain [3] gives a lower bound of $\Omega(\log n/\log\log n)$ for the worst-case distortion of any contraction of n-point metric spaces into ℓ_2. From Theorem A.3 we get the following improvement over Bourgain's lower bound.

COROLLARY A.4. *Any contraction of $\mathcal{M}_n$ into ℓ_2 has distortion $\Omega(\log n)$.*

Proof. Any finite Euclidean space is ℓ_1-embeddable. (It is well known that L_2 isometrically embeds in L_1, see, e.g., [4]. For finite metric spaces, L_1-embeddability implies ℓ_1 embeddability; see [6].) □

Remark. The question of improving Bourgain's lower bound was raised by Linial, London, and Rabinovich. A similar theorem, derived independently, is shown in [21, 22].

REFERENCES

[1] G. M. ADELSON-WELSKY, E. A. DINITS, AND A. V. KARZANOV, *Flow Algorithms*, Nauka, Moscow, 1975 (in Russian).

[2] D. AVIS AND M. DEZA, *The cut cone, L^1-embeddability, complexity and multicommodity flows*, Networks, 21 (1991), pp. 595–617.

[3] J. BOURGAIN, *On Lipschitz embedding of finite metric spaces in Hilbert space*, Israel J. Math., 52 (1985), pp. 46–52.

[4] J. BRETAGNOLLE, D. DACUNHA-CASTELLE, AND J. L. KRIVINE, *Lois stables et espaces L^p*, Ann. Inst. H. Poincaré, 2 (1966), pp. 231–259.

[5] E. DAHLHAUS, D. S. JOHNSON, C. H. PAPADIMITRIOU, P. D. SEYMOUR, AND M. YANNAKAKIS, *The complexity of multiway cuts*, in Proc. of the 24th Annual ACM Symposium on Theory of Computing, ACM, New York, 1992, pp. 241–251.

[6] M. DEZA AND M. LAURENT, *Applications of Cut Polyhedra*, Technical report BS-R9221, CWI, Amsterdam, 1992.

[7] P. ELIAS, A. FEINSTEIN, AND C. E. SHANNON, *A note on the maximum flow through a network*, IRS Trans. Inform. Theory, 2 (1956), pp. 117–119.

[8] L. R. FORD AND D. R. FULKERSON, *Maximal flow through a network*, Canadian J. Math., 8 (1956), pp. 399–404.

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

[10] N. GARG, *A Deterministic $O(\log k)$-Approximation Algorithm for Sparsest-Cut*, 1995, preprint.

[11] N. GARG, V. V. VAZIRANI, AND M. YANNAKAKIS, *Primal-dual approximation algorithms for integral flow and multicut in trees, with applications to matching and set cover*, in Proc. 20th International Colloquium on Automata, Languages, and Programming, (ICALP '93), Springer-Verlag, New York, 1993, pp. 64–75.

[12] N. GARG, V. V. VAZIRANI, AND M. YANNAKAKIS, *Approximate max-flow min-(multi)cut theorems and their applications*, in Proc. of the 25th Annual ACM Symposium on Theory of Computing, ACM, New York, 1993, pp. 698–707.

[13] M. X. GOEMANS AND D. P. WILLIAMSON, *.878-approximation algorithms for MAX CUT and MAX 2SAT*, in Proc. of the 26th Annual ACM Symposium on Theory of Computing, ACM, New York, 1994, pp. 422–431.

[14] T. C. HU, *Multicommodity network flows*, Oper. Res., 11 (1963), pp. 344–360.

[15] D. KARGER, R. MOTWANI, AND M. SUDAN, *Approximate graph coloring by semidefinite programming*, in Proc. of the 35th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 2–13.

[16] P. Klein, A. Agrawal, R. Ravi, and S. Rao, *Approximation through multicommodity flow*, in Proc. of the 31st Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 726–737.

[17] P. Klein, S. Plotkin, and S. Rao, *Excluded minors, network decomposition, and multicommodity flow*, in Proc. of the 25th Annual ACM Symposium on Theory of Computing, ACM, New York, 1993, pp. 682–690.

[18] B. Korte, L. Lovász, H. J. Promel, and A. Schrijver, eds., *Paths, Flows, and VLSI-Layout*, Springer-Verlag, New York, 1990.

[19] F. T. Leighton, F. Makedon, S. Plotkin, C. Stein, É. Tardos, and S. Tragoudas, *Fast approximation algorithms for multicommodity flow problems*, in Proc. of the 23rd Annual ACM Symposium on Theory of Computing, ACM, New York, 1991, pp. 101–111; J Comput. System Sci., 50 (1995), pp. 228–243.

[20] F. T. Leighton and S. Rao, *An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms*, in Proc. of the 29th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 422–431.

[21] N. Linial, E. London, and Y. Rabinovich, *The geometry of graphs and some of its algorithmic applications*, in Proc. of the 35th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 577–591.

[22] N. Linial, E. London, and Y. Rabinovich, *The geometry of graphs and some of its algorithmic applications*, Combinatorica, 15 (1995), pp. 215–245.

[23] M. Lomonosov, *Combinatorial approaches to multiflow problems*, Discrete Appl. Math., 11 (1985), pp. 1–93.

[24] G. A. Margulis, *Explicit constructions of concentrators*, Problems Inform. Transmission, 9 (1975), pp. 325–332.

[25] J. Matoušek, *Note on bi-Lipschitz embeddings into normed spaces*, Commentationes Mathematicae Univ. Carolinae, 33 (1992), pp. 51–55.

[26] H. Okamura and P. D. Seymour, *Multicommodity flows in planar graphs*, J. Combin. Theory Ser. B, 31 (1989), pp. 75–81.

[27] S. Plotkin and É. Tardos, *Improved bounds on the max-flow min-cut ratio for multicommodity flows*, in Proc. of the 25th Annual ACM Symposium on Theory of Computing, ACM, New York, 1993, pp. 691–697.

[28] B. Rothschild and A. Whinston, *On two-commodity network flows*, Oper. Res., 14 (1966), pp. 377–387.

[29] P. D. Seymour, *Four terminous flows*, Networks, 10 (1980), pp. 79–86.

[30] P. D. Seymour, *Matroids and multicommodity flows*, European J. Combin., 2 (1981), pp. 257–290.

[31] F. Shahrokhi and D. W. Matula, *The maximum concurrent flow problem*, J. ACM, 37 (1990), pp. 318–334.

[32] S. Tragoudas, *VLSI Partitioning Approximation Algorithms Based on Multicommodity Flow and Other Techniques*, Ph.D. thesis, University of Texas at Dallas, 1991.

[33] J. H. Wells and L. R. Williams, *Embeddings and Extensions in Analysis*, Springer-Verlag, New York, 1975.

[34] Y. Ye, *An $O(n^3L)$ potential reduction algorithm for linear programming*, Math. Programming, 50 (1991), pp. 239–258.

SIAM J. COMPUT.
Vol. 27, No. 1, pp. 302–316, February 1998
© 1998 Society for Industrial and Applied Mathematics
013

A SUBLINEAR TIME DISTRIBUTED ALGORITHM FOR MINIMUM-WEIGHT SPANNING TREES*

JUAN A. GARAY†, SHAY KUTTEN‡, AND DAVID PELEG§

Abstract. This paper considers the question of identifying the parameters governing the behavior of fundamental *global* network problems. Many papers on distributed network algorithms consider the task of optimizing the running time successful when an $O(n)$ bound is achieved on an n-vertex network. We propose that a more sensitive parameter is the network's diameter $Diam$. This is demonstrated in the paper by providing a distributed minimum-weight spanning tree algorithm whose time complexity is sublinear in n, but linear in $Diam$ (specifically, $O(Diam + n^{\varepsilon} \cdot \log^* n)$ for $\varepsilon = \frac{\ln 3}{\ln 6} = 0.6131...$). Our result is achieved through the application of graph decomposition and edge-elimination-by-pipelining techniques that may be of independent interest.

Key words. MST, min-weight spanning trees, distributed algorithms

AMS subject classifications. 05C05, 05C85, 68Q22, 68Q25, 68R10

PII. S0097539794261118

1. Introduction.

1.1. Motivation. In many papers on distributed network algorithms, the task of optimizing the running time is considered successful when an $O(n)$ bound is achieved on an n-vertex network. Typically, the justification is that there exist n-vertex graphs for which this bound is the best possible. The sequence of solutions to the leader election problem (LE) exemplifies the reasoning above. Following the $O(n \log n)$ running time and a first improvement by Chin and Ting [CT] and Gafni [G] (with an $O(n \log^* n)$ running time), Awerbuch gave an "optimal" $O(n)$-time solution to the problem [A1]. Again, this solution is optimal in the sense that there exist networks for which this is the best possible.

This type of optimality may be thought of as "existential" optimality; namely, there are points in the class of input instances under consideration for which the algorithm is optimal. A stronger type of optimality, which we may analogously call "universal" optimality, occurs when the proposed algorithm solves the problem optimally on *every* instance.

An interesting "side effect" of universal optimality is that a universally optimal algorithm precisely identifies the parameters of the problem that are inherently responsible for its complexity. For example, returning to the LE problem, a more careful

* Received by the editors January 3, 1994; accepted for publication (in revised form) February 5, 1996. Based on *A sub-linear time distributed algorithm for minimum-weight spanning trees* (preliminary version), by J. A. Garay, S. Kutten, and D. Peleg, which appeared in the *Proceedings of the 34th Annual IEEE Symposium on the Foundations of Computer Science,* Palo Alto, CA, IEEE Computer Society Press, Los Alamitos, CA, 1993, pp. 659–668.

http://www.siam.org/journals/sicomp/27-1/26111.html

† IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598 (garay@ watson.ibm.com). Part of this author's work was done while visiting the Weizmann Institute of Science, Rehovot, Israel.

‡ IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598 (kutten@ watson.ibm.com).

§ Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, 76100, Israel (peleg@wisdom.weizmann.ac.il). The work of this author was supported in part by a Walter and Elise Haas Career Development Award and by a grant from the Basic Research Foundation. Part of the work was done while visiting IBM T.J. Watson Research Center, Yorktown Heights, NY.

look reveals that the inherent parameter is the network's diameter $Diam$. Indeed, it was observed in [P] that it is possible to give a trivial $O(Diam)$-time distributed LE algorithm (although it should be noted that the solutions mentioned above were also message-optimal, whereas the algorithm of [P] is not).

The interesting question that arises is, therefore, whether it is possible to identify the inherent graph parameters associated with the distributed complexity of various fundamental network problems and to develop universally optimal algorithms for them.

A closely related question has been dealt with before in the context of studying the role of *locality* in distributed computing. Various problems were shown to be essentially local and hence amenable to a localized algorithm with very fast (e.g., polylogarithmic) running times. Notable examples include computing maximal independent sets and graph coloring [GPS, L, AGLP, PS2]. Locality-based techniques were also developed for reducing communication and time complexities for other problems whose local natures are less apparent [AP1, AP2, AKP].

In contrast, we are interested here in problems that are essentially *global*, i.e., ones that do not admit localized solutions but rather always require the algorithm to "traverse" the network. Problems of this type still raise the interesting (if more modest) question of deciding whether $\Omega(n)$ time is essential—or is the network's *diameter* the inherent parameter? In the latter case, it would be desirable to devise algorithms for these network problems that have a better complexity for the case of graphs with low diameter.

In this paper, we tackle the classical minimum-weight spanning tree (MST) problem. This problem has been studied before as a canonical example for a graph-algorithmic problem whose communication-efficient distributed solution poses some surprisingly nontrivial subtleties [GHS]. The time complexity of the algorithm of [GHS] is $O(n \log n)$, which was later improved to the (existentially) "optimal" $O(n)$ in [A1]. As with other problems, such as the above LE example, it is natural to ask whether $O(n)$ is universally optimal or if it can be improved.

Once again, the MST problem proves to be a worthy candidate for this type of study. In other tree constructions, such as the breadth-first-search (BFS) tree (which is closely related to the LE problem), it is intuitively clear that the true time bound should be related to the network's diameter $Diam$, since the depth of the constructed tree is proportional to $Diam$. In contrast, the MST of a given network may be considerably deeper than $Diam$, and in fact, may be as high as $\Omega(n)$. Hence construction methods based on communication on the tree structure itself are doomed to require $\Omega(n)$ time, and the problem of breaking the $\Omega(n)$ barrier seems intrinsically harder.

In this paper we get closer to identifying the inherent parameters governing the behavior of distributed MST construction by presenting a distributed MST algorithm whose time complexity is sublinear in n, and linear in $Diam$ (specifically, $O(Diam + n^{\varepsilon} \cdot \log^* n)$ for $\varepsilon = \frac{\ln 3}{\ln 6} = 0.6131...$), thus breaking the $O(n)$ barrier. This result is achieved through the application of graph decomposition and edge elimination techniques that may be interesting in their own right.

1.2. Model and definitions. In this paper we focus on the problem of devising a time-efficient distributed MST algorithm. The statement of the problem is as follows. The network is represented by an undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of links between them. (Henceforth we use the terms "graph" and "network" interchangeably, when no ambiguity arises.) The graph

is given with a weight function $\omega : E \to R^+$ on the edges, such that each node in V is associated with its own processor, and processors are able to communicate with each other via the edges in E. The goal is to have the nodes (processors) cooperate to construct a tree covering the nodes in V whose total edge weight is no greater than any other spanning tree for G.

We assume that nodes have unique identifiers and that each edge $e \in E$ is associated with a distinct weight $\omega(e)$ known to the adjacent nodes. The usefulness of having distinct edge weights stems from the fact that this property guarantees that the MST is unique. Clearly, having distinct weights is not an essential requirement, since one can always "create" them by appending the adjacent node's numbers to them. However, it is known that if the graph has neither distinct edge weights nor distinct node identifiers, then no distributed algorithm exists for computing an MST with a bounded number of messages [GHS].

For every subgraph F of the network, let $Diam(F)$ denote the diameter of F, i.e., the maximum distance between any two vertices of F, where distance is measured in the unweighted sense, i.e., in the number of hops.

In order to be able to concentrate on the central issue of time complexity, we shall follow the common trend of stripping away nonessential complications. In particular, we ignore the communication cost of our algorithm, i.e., the number of messages it uses. (We comment that while this number is not optimized in any way in our solution as presented in the current paper, it is not very large. In particular, it is still considerably smaller than the number of messages needed for every node to learn the topology of the entire network, as is used in a number of existing routing schemes, e.g., Internet routing [MRR].)

We also assume that the computation performed by the network is *synchronous*; namely, computation proceeds in rounds, governed by a global clock, with each round taking one time unit. In each round each processor can examine the messages sent to it by its neighbors (if any), compute, and send messages to any subset of its neighbors. Such messages, if sent, are available to their recipients in the next round. The *time complexity* of a synchronous algorithm is the number of time units elapsing until the termination of the algorithm. This assumption is quite common in the literature. Note, however, that here it is not essential, since our decision to ignore communication costs allows us to freely use a synchronizer of our choice; for example, synchronizer α [A2] enables an asynchronous network to run any protocol that was designed for synchronous networks, with the same time complexity, at the cost of some increase in the message complexity.

Still, we shall not adopt the extreme model employed in previous studies of locality issues (cf. [L]), in which messages of arbitrary size are allowed to be transmitted in a single time unit, since in this model the refined distinctions we focus on here disappear. Clearly, if unbounded-size messages are allowed, then the problem can be trivially solved in time $O(Diam(G))$ by collecting the entire graph's topology into a central node, computing an MST locally, and broadcasting the result throughout the network.

Consequently, we will assume the more realistic (and rather common) model in which messages have size $O(\log n)$, and a node may send at most one message on each edge at each time unit.

We will also make the assumption that edge weights are polynomial in n, so an edge weight can be sent in a single message. (This assumption is required for the time analysis of all previous algorithms as well [GHS, A1].)

1.3. Our results. The original distributed MST algorithm of Gallager, Humblet, and Spira [GHS] has a time complexity of $O(n \log n)$. This was later improved by Awerbuch, who gave an $O(n)$-time algorithm [A1]. In this paper, we present a distributed MST algorithm with time complexity $O(Diam(G) + n^{\varepsilon} \cdot \log^* n)$ for $\varepsilon = \frac{\ln 3}{\ln 6} = 0.6131...$.

2. Overview of the MST algorithm. Our algorithm is based on a careful combination of two distinct approaches to the (distributed) construction of an MST. Thus, before explaining our algorithm, it is instrumental to review the two approaches and try to understand their shortcomings (when used individually).

2.1. Distributed growth approach. The first approach is the construction method that is at the basis of the MST algorithm of [GHS] (referred to from now on as the GHS algorithm). This algorithm does not utilize a central controller, or "center of activity," but rather allows a large number of processes to proceed simultaneously and independently in the network. These processes gradually *grow* the MST from scratch.

The GHS algorithm operates by growing so-called *fragments* in a distributed manner, with each fragment consisting of a portion of the final MST. In each iteration of the algorithm, the nodes of each fragment explore the immediate neighborhood of the fragment and collectively decide on a neighboring fragment to merge with (by adding the connecting edge to the tree), thus creating a larger fragment of the final MST. This is a distributed version of the "blue" rule for MST construction (cf. [T, p. 71]).

It is crucial to understand why algorithms based on this approach cannot guarantee a time complexity proportional to $Diam(G)$. The inherent difficulty lies in the fact that the communication necessary for making the merging decisions for each fragment is done *on the fragment itself*. This is a problem, since the MST is not guaranteed to have depth proportional to $Diam(G)$, and neither are any of its fragments. In fact, it is easy to come up with examples for n-vertex graphs with diameter 1 whose MST has depth $n - 1$. Thus, any approach based on communicating over the MST itself will have time complexity proportional to n in the worst case on some graphs.

Note, however, that in the initial stages of the GHS algorithm, the fragments are still small, and therefore communication on them is not as expensive. The idea on which our algorithm is based is thus to start by running the GHS algorithm up to an appropriately chosen point, and then switch to a different algorithm. In order for this idea to work, it is essential to have a version of the GHS algorithm that controls the rate of growth of the different fragments, preventing some fragments from growing too large while other are still very small. (The original algorithm of [GHS] allows uncontrolled growth of fragments.)

2.2. Coordinated elimination approach. The second approach to the distributed construction of an MST is based on synchronous, coordinated, centralized operation. One extreme example for an algorithm operating in this way is the centralized algorithm mentioned in the introduction, in which the entire graph's topology is collected into a central node, which then computes an MST locally and broadcasts the result throughout the network. This algorithm is slowed down considerably by the heavy communication bottlenecks that are bound to be created along the paths leading to the center node.

The crucial idea at the basis of the second approach is that some of this communication burden can be reduced by delegating the work of the center node to other

nodes along the paths leading to the center. This can be achieved if, instead of attempting to *build* the MST, we concentrate on *eliminating candidate edges*, using the so-called "red rule" for MST construction (cf. [T, p. 71]).

In order to control the complexity of the edge-elimination process, we employ a new efficient pipelining technique aimed at overcoming congestion. This new technique deserves some discussion, since it is different and considerably simplified compared to the one presented in an earlier version of this paper [GKP]. The new technique is motivated by the following complication in the solution of [GKP]. In the end of the first stage of the algorithm, nodes are informed about candidate edges in the graph (those that connect clusters). In the third stage of the [GKP] algorithm, the nodes forward the description of those edges to their parents on a spanning tree, toward the root. The problem is that there might be too many such edges, congesting the accumulation process on the tree. The algorithm of [GKP] overcomes this difficulty by introducing an intermediate stage employing a rather complex method (centered on locality-based ideas related to [ADDJ, AGPV, B, PS1]) for eliminating short cycles, and thus reducing the number of remaining edges for the third stage.

The algorithm presented here has only two stages. Both the second (cycle elimination) and the third stages of [GKP] are replaced by a single stage, similar to the third stage of [GKP], in which the nodes forward descriptions of candidate edges to their parents on a breadth-first-search spanning tree. However, we incorporate a new technique of cycle elimination into this stage. This new pipelining technique is very simple, and the main novelty lies in its analysis. As in the third stage of the algorithm in [GKP], nodes on the spanning tree forward the descriptions of their candidate edges to their parents toward the root. However, here a node avoids forwarding the description of *cycle-heavy* edges. That is, a node forwards the description of edges according to their weight and does not forward the description of an edge that closes a cycle with edges whose description has already been forwarded. Let us now give a hint regarding the difficulty in analyzing this stage of the protocol. It is not hard to prove that all non-MST edges get eliminated by this process. The difficulty lies in analyzing the time complexity. Note that the description above (i.e., that a node does not forward a description of a cycle-heavy edge) seems to suggest that at certain points in time a node might be forced to wait, due to not having an edge description that is eligible for forwarding. Thus, it may seem that the convergecast may not be fully pipelined, and hence may take a long time. We prove that this simple convergecast is fully pipelined, and thus its running time is the one required, eliminating the need for the complex cycle elimination stage of [GKP]. This pipelining proof may be of interest in itself.

2.3. Combined approach. Let us now outline the structure of our algorithm. The algorithm consists of two parts as follows.

`Algorithm Sublinear-`MST

Part I: `Controlled-GHS`;

Part II: Edge elimination.

As its name indicates, `Controlled-GHS` (part I) is a modified variant of the original algorithm of [GHS]. The purpose of the modification is to produce a balanced outcome in terms of number and diameter of the resulting fragments. This is achieved by computing, in each phase, a small dominating set on the fragment forest and merging

fragments accordingly. This, in turn, is achieved by invoking the distributed maximal independent set (MIS) algorithm of [GPS]. At the end of this phase, we are left with a "small" number of fragments, all of which have a "small" diameter.

Finally, part II performs the elimination of most of the remaining edges, leaving only a tree connecting the fragments, and thus yielding the final MST. This elimination process is carried out in a distributed manner and requires nodes to forward the description of certain edges to a central node, while eliminating the description of certain other edges. (The forwarding is done on a superimposed breadth-first tree, and the center is the root of this tree.) The central node does the final elimination, which yields the output MST tree.

The details of each part are given in the remainder of the paper, followed by the analysis of the total complexity.

3. Part I: Controlled-GHS. In this section we provide a modified, *controlled* version of the Gallager-Humblet-Spira algorithm for MST that is suitable for our purposes. We first provide a brief overview of the original algorithm of [GHS].

3.1. Brief description of the GHS algorithm. In the original distributed algorithm of [GHS], nodes form themselves into *fragments* of increasing size. Initially, all nodes are in singleton fragments. Nodes in each fragment F are connected by edges that form a *rooted* MST, $T(F)$, for the fragment. (Initially, the sole node comprising a fragment is also its root.) Each node (other than the root) in a fragment has a pointer to one of its neighbors, which is the next node on the path over the tree to the root; moreover, each node "knows" the root Id. (In the remainder of this paper we loosely use the word "fragment" to mean both the collection of nodes F and the corresponding tree $T(F)$.)

The events in the algorithm are divided into *phases*. Each phase takes as its input the fragment structure output by the previous one and outputs larger fragments. (The first phase takes as an input the singleton fragments, one per node.)

Let us now describe one phase. We present a simplified version of the GHS algorithm compared to the original algorithm of [GHS]. (The original version is more complex because of the desire to save messages and because of the asynchronous nature of the networks for which it was designed.)

Within each fragment F, nodes cooperate to find the minimum-weight *outgoing* edge in the entire fragment (an outgoing edge of a fragment F is an edge with one endpoint in F and another at a node outside it). The strategy for identifying this edge involves broadcasting over the fragment's tree $T(F)$, asking each node separately for its own minimum-weight outgoing edge. These edges are then sent upwards on the tree $T(F)$, toward the root. Each intermediate node first collects this information from all its children in the tree and then passes up only the lowest-weight edge it has seen (which is therefore the lowest-weight edge in its subtree). The minimum-weight outgoing edge is selected by the root to be included in the final MST.

Once a fragment's minimum-weight outgoing edge is found, a message is sent out over that edge to the *chosen* fragment on the other side. The two fragments then combine, possibly along with several other fragments, into a new, larger fragment. If the other fragment chose the same edge, then the two fragments agree at that point to combine, and the edge they both chose is termed the *core edge*. Otherwise, following the route from a fragment F_1, to its chosen fragment F_2, to F_2's chosen fragment F_3, and so on, [GHS] show that one must eventually reach two fragments that agree to combine over a core edge. The combined fragment includes these two, as well as all those other fragments that have such a route to them.

The combined fragment has a root; it is the node with the higher Id of the two endpoints of the core edge. (This can be simulated in the case that nodes do not have distinct Id's but edges have distinct weights.)

This concludes the description of a single phase. In the next phase, the new fragment finds its own minimum-weight outgoing edge, and the entire process is repeated until all the nodes in the graph have combined themselves into one single fragment. Each fragment (of size 2 or greater) is identified by the fragment's *core edge* and root.

We remark that in the original GHS algorithm, all nodes operate asynchronously. This creates the risk of undesirable "growth patterns" of fragments, resulting in excessive communication costs (measured in number of messages). This problem is handled by using special rules for merging fragments, designed to prevent these complications. These rules are based on a "balanced data structure" approach. A *phase number* is associated with each fragment. If $phase(F) = l$ for a given fragment F, then the number of nodes in F is greater than or equal to 2^l. Initially, all fragments (singleton nodes) are at phase 0. When two fragments at phase l are combined together, the resulting new fragment has phase $l + 1$. Thus, the total number of messages is kept to $O(n \log n)$ (although some more complex rules are needed to allow merges between fragments of unequal phases). Similarly, it is not hard to show by induction on the phase numbers that the time complexity of the algorithm is $O(n \log n)$ time units.

We refer the reader to [GHS] for further details.

3.2. Computing a small dominating set on a tree. In this subsection we present a procedure `Small-Dom-Set` for computing a small dominating set on a given tree. The procedure makes use of a subprocedure for computing a maximal independent set in the tree. (A set M of vertices in a tree T is said to *dominate* the tree if every vertex outside M has a neighbor in M.) A distributed version of procedure `Small-Dom-Set` will later be used as a component in our `Controlled-GHS` algorithm, which is a modification of GHS.

Our goal is as follows. Given a rooted tree T with a vertex set $V(T)$, find a set of vertices $M \subseteq V(T)$ such that

1. M dominates $V(T)$, and
2. $|M| \leq \frac{|V(T)|}{2}$.

Furthermore, we would like this procedure to be amenable to a fast distributed implementation.

The procedure is based on the following. For a vertex $v \in V(T)$, let $\texttt{Child}(v)$ denote the set of v's children in T. We use a *level* function $\check{L}(v)$ on the nodes, defined as follows:

$$\check{L}(v) = \begin{cases} 0 & \text{if } v \text{ is a leaf,} \\ 1 + \min_{u \in \texttt{Child}(v)}(\check{L}(u)) & \text{otherwise.} \end{cases}$$

We denote by $\check{L}_i$ the set of tree nodes at level i,

$$\check{L}_i = \{v \mid \check{L}(v) = i\}.$$

Procedure `Small-Dom-Set` for computing a dominating set M on a tree T is presented next.

`Algorithm Small-Dom-Set`

1. Mark the nodes of T with level numbers $\check{L}(v) = 0, 1, 2$;
2. Select an MIS, Q, in the set R of unmarked nodes;
3. $M \leftarrow Q \cup \check{L}_1$.

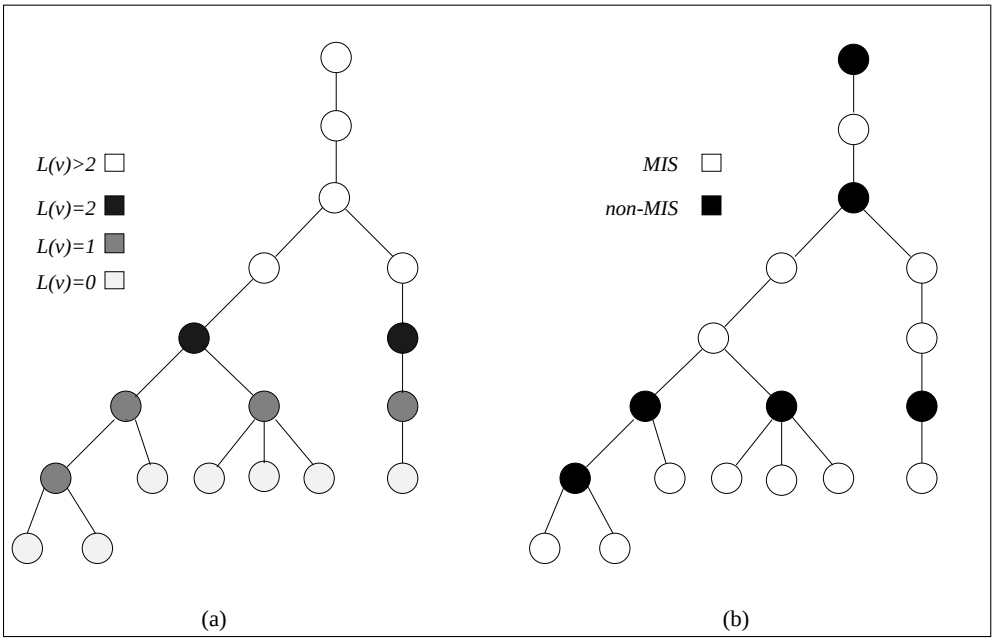


FIG. 1. (a) *The level numbers marked by the algorithm on a given tree* T. (b) *A small dominating set* M *on the tree* T.

A pictorial example is given in Figure 1.

For the distributed implementation discussed in the next subsection, it is important to note that although the level numbers $\check{L}(v)$ are defined for every vertex v in the tree, only the vertices belonging to the first three levels, $\check{L}_0$, $\check{L}_1$, and $\check{L}_2$, are actually marked.

The fact that procedure `Small-Dom-Set` produces a dominating set is established by the following lemma.

LEMMA 3.1. *Let* M *be the outcome of procedure* `Small-Dom-Set` *on the tree* T. *Then for every node* $v \notin M$ *there exists an adjacent node* $v' \in M$.

Proof. Partition $V(T)$ into $\check{L}_0 \cup \check{L}_1 \cup \check{L}_2 \cup R$. The set M output by procedure `Small-Dom-Set` is composed of $Q \cup \check{L}_1$. Now, by choice of Q, it dominates each node of $R \setminus Q$. Also, each node of $\check{L}_0 \cup \check{L}_2$ has a neighbor in $\check{L}_1$. □

The "smallness" of the resulting dominating set is guaranteed by the following lemma.

LEMMA 3.2. $|M| \leq \frac{|V(T)|}{2}$.

Proof. By construction, $M = \check{L}_1 \cup Q$. It is clear that $|\check{L}_1| \leq |\check{L}_0|$, and hence

$$|\check{L}_1| \leq \frac{|\check{L}_0 \cup \check{L}_1|}{2} \ . \tag{1}$$

We now claim that

$$|Q| \leq \frac{|R \cup \check{L}_2|}{2} \ . \tag{2}$$

This can be proved by selecting for every $v \in Q$ a *distinct* match $\omega(v) \in (R \cup \check{L}_2) - Q$, thus establishing that $|R \cup \check{L}_2| \geq 2 \cdot |Q|$. The matching is done as follows: pick $\omega(v)$ to

be an arbitrary child of v (by definition, vertices in Q always have children in $R \cup \check{L}_2$). Distinctness is guaranteed by the fact that each node in the tree has a unique parent. It now follows from (1) and (2) that

$$|M| = |Q \cup \check{L}_1| \leq \frac{|R \cup \check{L}_2|}{2} + \frac{|\check{L}_0 \cup \check{L}_1|}{2} = \frac{|V(T)|}{2}. \qquad \square$$

3.3. Controlled-GHS.

3.3.1. Description of the `Controlled-GHS` procedure. In this subsection, we provide the modified, *controlled* version of GHS (named `Controlled-GHS`) that is able to achieve the following:

1. Upon termination, the number of fragments is bounded from above by N (for N to be specified later).
2. Throughout the execution, the diameter of every fragment F satisfies $Diam(F) \leq d$ (for d to be specified later).

Intuitively, since we focus on a synchronous algorithm, and we do not care about communication complexity, our version of GHS is simpler than the original algorithm. In particular, we are oblivious to balancing fragment sizes, and we do not need to use the phase rules used in the original algorithm, since phases are imposed by phase synchronization (using the assumed global clock).

More specifically, `Controlled-GHS` starts with singleton fragments, just like GHS (see subsection 3.1), and executes a total of I phases. Each phase outputs a collection of fragments that serve as the input for the next phase. A fragment is a rooted tree, where each node (except for the root) has a pointer to the edge that leads to its parent in the tree. Each phase of `Controlled-GHS` consists of the following two stages.

Stage 1. Consider the fragments that are input to the phase. Execute a phase of GHS up to a point where each such input fragment F has chosen its minimum-weight outgoing edge, i.e., has decided with which other fragment in the current fragment collection it wants to merge.

This decision induces a "forest" structure on the fragment collection (possibly with length-2 loops at the tree roots). Henceforth we refer to this structure as the *fragment forest*, denoted FF.

Stage 2. Break the resulting trees into "small" ($O(1)$ depth) trees, and merge only these small trees.

This process is depicted in Figure 2. To accomplish Stage 2, the algorithm first computes a dominating set $M_{FF}(\tilde{T})$ on each tree $\tilde{T}$ of the fragment forest FF. Note that the nodes of this tree are input fragments of the current phase of the algorithm. Let M_{FF} be the union over the trees $\tilde{T}$ (of the FF forest) of $M_{FF}(\tilde{T})$. The algorithm then lets each fragment $F \notin M_{FF}$ pick one neighboring fragment $F' \in M_{FF}$ and merge with it. This causes the actual merges performed in a phase of `Controlled-GHS` to have the form of "stars" in the fragment forest FF, and prevents merges along long chains, hence bounding the diameter of the resulting fragments.

The dominating sets are computed using a distributed implementation of procedure `Small-Dom-Set`, denoted `Dist-SDS`, applied separately to each tree $\tilde{T}$ in the fragment forest FF. A key aspect in this computation is the use of a distributed algorithm for computing a maximal independent set. Most distributed MIS algorithms in the literature (e.g., [AGLP, PS2]) can be used for our purposes. In fact, our solution only requires to compute an MIS on a tree, so we can use a distributed version of the algorithm of [GPS], which is optimal for trees. (This algorithm makes use of $O(\log n)$-bit messages and therefore can be used within our model.)

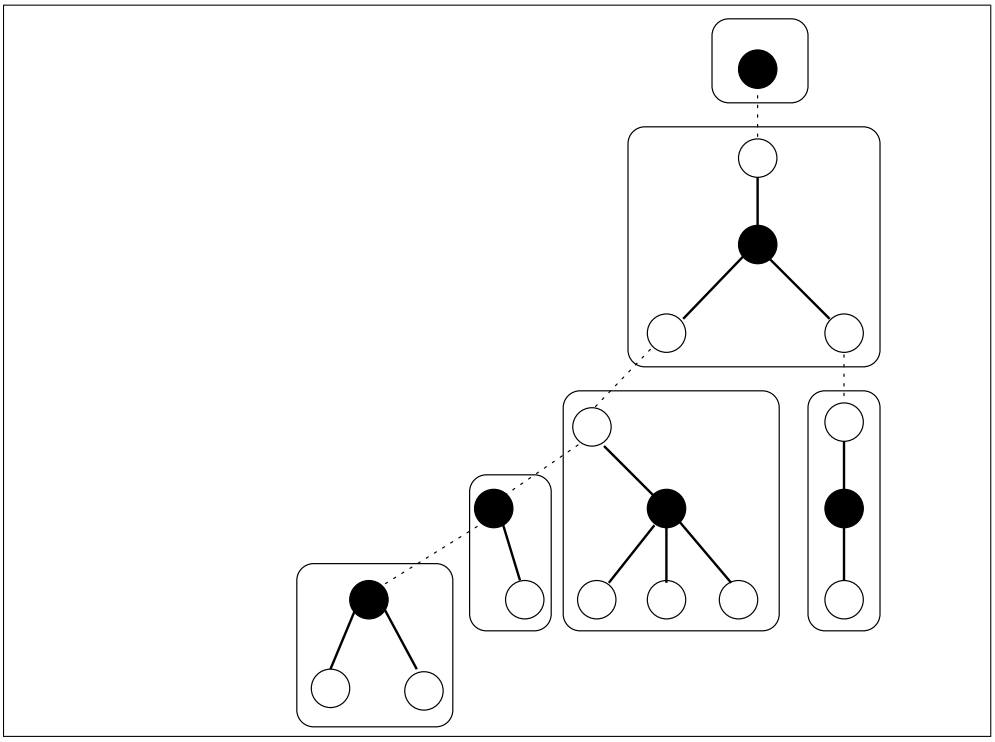


FIG. 2. *A phase of* `Controlled-GHS`*: the tree T of Figure* 1b *is broken into "small" trees (represented by the solid edges) according to the MIS.*

The algorithm of [GPS] (henceforth referred to as the `GPS` algorithm) has the following properties.

LEMMA 3.3 (see [GPS]). *Applied in a given n-vertex tree G, Algorithm* `GPS` *computes (in a synchronous manner) a maximal independent set (MIS) of G using $O(\log n)$-bit messages, and its time complexity is $O(\log^* n)$.*

Note that although procedure `Dist-SDS` is applied to the trees of the fragment forest FF, it is actually executed on the original network itself. Hence the procedure operates by simulating each fragment by a single representative, say, its root.

3.3.2. Analysis of controlled-GHS. The bounds on the number and diameter of fragments are established by the next lemma. Given a fragment forest FF, let $Diam(FF) = \max_{F \in FF}\{Diam(F)\}$. Let FF_i denote the fragment forest produced by the algorithm `Controlled-GHS` at the end of the ith phase. (FF_0 is the initial set of vertices.)

LEMMA 3.4. *In each phase i of* `Controlled-GHS`*,*

1. *the number of fragments at least halves, i.e., $|FF_i| \leq |FF_{i-1}|/2$, and*
2. *the maximum diameter of a fragment increases by a factor of at most* 3 (*possibly plus* 2)*, i.e., $Diam(FF_i) \leq 3Diam(FF_{i-1}) + 2$.*

Proof. The number of new fragments in each tree $T = T(F)$ of the fragment forest FF at the end of a phase is equal to $|M(T)|$. By Lemma 3.2, $|M(T)| \leq |V(T)|/2$. Hence the same holds for the entire fragment forest, and claim 1 of the lemma follows. Claim 2 is readily satisfied since the merges are star shaped. □

COROLLARY 3.5. *After running* `Controlled-GHS` *for I phases,*

1. *the number of fragments in* FF_I *is at most* $N(I) = \frac{n}{2^I}$, *and*
2. $Diam(FF_I) \leq d(I) = 3^I - 1$ *for every fragment* F.

Let us now turn to analyzing the time complexity of `Controlled-GHS`. We first examine the performance of procedure `Dist-SDS`.

LEMMA 3.6. *When executed on a fragment forest* FF, *procedure* `Dist-SDS` *takes time* $O(\log^* n) \cdot Diam(FF)$.

Proof. It is easy to see that if a fragment F is a leaf in the fragment forest FF, then it identifies itself as such, and subsequently marks itself $\check{L}_0$, in time $O(Diam(F))$. Similarly, fragments mark themselves $\check{L}_1$ and $\check{L}_2$ in time $O(Diam(F))$ as well. Another part that affects the time complexity is the MIS computation. By Lemma 3.3, the time complexity of the `GPS` algorithm is $O(\log^* n)$. Note that since the procedure is executed on the original network itself, it is slowed down by a factor of $O(Diam(FF))$. Hence the implementation of the `GPS` algorithm on the fragments of FF is slowed down to $O(\log^* n) \cdot Diam(FF)$ in our case. □

The properties of the `Controlled-GHS` algorithm are now summarized by the following *graph decomposition* lemma.

LEMMA 3.7. *When Algorithm* `Controlled-GHS` *is activated for* I *phases, it takes* $O(3^I \cdot \log^* n)$ *time and yields a fragment forest* FF_I *of up to* $N(I) = n/2^I$ *fragments, of diameter* $Diam(FF_I) \leq d(I) = 3^I - 1$. *Each fragment in the forest is a fragment of an MST of the graph* G.

Proof. By Lemma 3.6, each phase i, $1 \leq i \leq I$, of `Controlled-GHS` takes time at most $Diam(FF_{i-1}) \cdot O(\log^* n)$. By Corollary 3.5, this is at most $3^{i-1} \cdot O(\log^* n)$. Thus, the total time is given by

$$\sum_{i \leq I} (3^{i-1} \cdot O(\log^* n)) \leq 3^I \cdot O(\log^* n) \ .$$

The size and diameter properties of the resulting fragment forest follow directly from Corollary 3.5. □

The rather even graph decomposition obtained by Algorithm `Controlled-GHS` may conceivably be useful for other purposes as well.

4. Part II: Edge elimination. The second part of our algorithm starts at the point where we are given a fragment graph $\tilde{F}_l$, whose vertex set $V(\tilde{F}_l)$ is a fragment forest FF_I containing $N \leq N(I)$ fragments of the MST, and whose edge set $E(\tilde{F}_l)$ is a collection of interfragment edges, which are the remaining candidates for joining the MST. We now proceed to reduce the total number of remaining interfragment edges (to the necessary $N - 1$).

4.1. The pipeline algorithm. We need the following technical definition. For a set of edges Q and a cycle-free subset $U \subseteq Q$, define $Cyc(U, Q)$ as the set of all edges $e \in Q \setminus U$ such that $U \cup \{e\}$ contains a cycle.

We are now ready to describe our pipelined procedure, given in Figure 3.

4.2. Analysis. Our analysis hinges on two main properties of the procedure. First, the edges reported by each intermediate node to its parent in the tree are sent in nondecreasing weight order. Second, each intermediate node transmits edges upwards in the tree *continuously* until it exhausts all the reportable edges from its subtree; namely, once the set of candidates RC is empty, the node will learn of no more reportable edges.

Let us first make the following straightforward but crucial observation.

1. Build a *breadth-first-search* tree B on G, with a root $r(B)$.
2. Throughout the execution, each node v on the tree maintains a set Q of all the interfragment edges it knows of (either directly or by "learning" from its children in the previous rounds), ordered by nondecreasing edge weights. Initially this set contains only the interfragment edges adjacent to it. It also maintains a set U (initially empty) of all the edges it has already sent up to its parent in the tree.
3. A leaf v starts sending edges upwards at pulse 0. An intermediate node v starts sending at the first pulse after it has received messages from all its children.
4. At each pulse i, the node sends up to its parent in the tree the lightest edge e in Q that has not been sent up in the previous rounds up to round $i-1$ (i.e., $e \notin U$), and which does not close a cycle with the edges of U. If no such edge exists then terminate participation in the pipelining part of the procedure (i.e., go to step 6).
   More formally, the node computes the set of *remaining candidates (RC)*
   $$RC = Q \setminus (U \cup Cyc(U, Q)).$$
   If $RC = \emptyset$, then v sends a "terminating" message to its parent in the tree and terminates its participation in the pipelining part of the protocol; else, it sends up to its parent the lightest edge e in RC.
5. The root $r(B)$ computes locally the set S of the $N-1$ edges participating in the MST of the fragment graph $\tilde{F}$ from among the edges it hears of from its children.
6. The root $r(B)$ then broadcasts (over the tree B) the resulting set of $N-1$ interfragment edges (constituting the MST for the fragment graph $\tilde{F}$) to all nodes in G.

FIG. 3. *Procedure* `Pipeline`.

LEMMA 4.1. *The edges reported by each intermediate node to its parent in the tree form a forest.*

Proof. This follows immediately from the rule used by the procedure to select the next edge to be transmitted upwards. □

LEMMA 4.2. *Every node v starts sending messages upwards at pulse $\hat{H}(v)$, where $\hat{H}(v)$ is the* height *function defined as follows:*

$$\hat{H}(v) = \begin{cases} 0 & \text{if } v \text{ is a leaf;} \\ 1 + \max_{u \in \texttt{Child}(v)}(\hat{H}(u)) & \text{otherwise.} \end{cases}$$

Proof. The proof is straightforward by induction on the tree structure, from the leaves upward. □

Our main technical lemma concerns the properties of a node on the tree in some round of the algorithm. Consider an intermediate node v at height H that has still

not terminated its participation in the algorithm, at round t, for some $t \geq H$. Note that each of the children of v in the tree is of height $H-1$ or lower, hence, by Lemma 4.2, all of them started transmission at round $H-1$ or earlier. Call a child *active* if it has not terminated yet (i.e., it has upcast an edge to v on round $t-1$). Let

$$A_t(v) = \{v_1, \ldots, v_p\}$$

be the set of v's active children at round t.

LEMMA 4.3.

(a) *At the beginning of round t, the candidate set RC examined by v contains at least one candidate edge upcast by each of its active children from $A_t(v)$.*

(b) *If v upcasts an edge of weight ω at round t, then all of the edges which v was informed of at round $t-1$ by its active children were of weight ω or greater.*

(c) *If v upcasts an edge of weight ω at round t, then any later edge it will learn of is of weight ω or greater.*

(d) *Node v upcasts edges in nondecreasing weight order.*

Proof. We prove the lemma by induction on the height of the tree, starting from the leaves upwards.

A *leaf* v has no (active or other) children, and therefore claims (a), (b), and (c) hold vacuously. Claim (d) follows trivially from the rules of the procedure.

Let us now consider an intermediate node v, and assume that the claims hold for each of its children. We need to prove the four claims for v. We start with claim (a).

Let U be the set of m edges upcast by v during the first $m = t - H$ rounds it has participated in (namely, rounds $H, \ldots, t-1$ if $t > H$). By Lemma 4.1, U forms a forest in G. Consequently, break U into the trees $U_1, \ldots, U_\ell$ in G, with $x_i = |U_i|$, where each such tree U_i has a vertex set $V(U_i)$ of exactly $x_i + 1$ vertices, and

$$\sum_i x_i = |U| = m. \tag{3}$$

Consider an active child u of v. Denote by D the set of edges upcast by u so far (up to and including round $t-1$). Since u was still active on round $t-1$, it has transmitted continuously to v since round $\hat{H}(u)$, which, as discussed before, is at most $H-1$. Therefore, we conclude that

$$|D| \geq m + 1. \tag{4}$$

Suppose, for the sake of contradiction, that none of the edges upcast by u is a candidate in round t. In other words, for each edge $e \in D$, either e was upcast by v earlier (namely, $e \in U$), or e closes a cycle with the edges of U (namely, $e \in Cyc(U)$). Thus, every such edge e has both of its endpoints in U. Furthermore, e cannot possibly connect endpoints that belong to two different trees U_i and U_j (since in that case, e would have been in neither U nor $Cyc(U)$).

This implies that the set D can be partitioned into sets $D_1, \ldots, D_\ell$ such that all the edges of D_i are restricted to vertices $V(U_i)$ of the tree U_i, for $1 \leq i \leq \ell$. Moreover, notice that each such set D_i is a forest, since the entire set D is a forest by Lemma 4.1.

The last two facts combined imply that

$$|D_i| \leq |V(U_i)| - 1 = x_i. \tag{5}$$

Combining (3), (4), and (5) we get

$$m+1 \leq |D| = \sum_i |D_i| \leq \sum_i x_i = m,$$

which is a contradiction. Hence claim (a) must hold.

Next we prove claim (b) as follows. Consider any active child u of v. Let e be the edge upcast by u on round $t-1$. (Note that e does not necessarily have to be in RC.) Let e' be some edge that was upcast by u at some round $t' \leq t-1$ and is still in the candidate set RC on round t (such an edge must exist by claim (a)). By the inductive hypothesis of claim (d), $\omega(e) \geq \omega(e')$. By the edge selection rule of node v, $\omega(e') \geq \omega$, and claim (b) follows.

Next we note that claim (c) follows trivially from claim (b). Finally, claim (d) follows trivially from claim (c) and the edge selection rule of the procedure. □

Finally, we need to argue that nodes do not terminate the algorithm prematurely.

LEMMA 4.4. *After a node v has terminated its participation in the algorithm, it will learn of no more reportable edges.*

Proof. We need to argue that once the set RC becomes empty, no new candidate edges will become known to v. We prove this fact by induction on the structure of the tree, starting from the leaves upward. The inductive step follows directly from claim (a) of Lemma 4.3, which guarantees that if RC is empty on round t, none of v's children have upcast it an edge in round $t-1$, and hence all of them have already terminated. □

LEMMA 4.5. *The running time of procedure* `Pipeline` *is bounded by $O(N + Diam(G))$, and its output is an MST for G.*

Proof. The fact that the resulting tree is an MST follows from the fact that the trees constructed in the first stage were fragments of the MST, and from the correctness of the "red rule" employed for edge elimination in the procedure (cf. [T, p. 71]). (Essentially the red rule says that an edge that is the heaviest on any cycle is not a part of any MST.)

As for the running time, the bound is derived from the following facts. First, the root of the tree receives at most N edges from each of its children. Second, the children send these edges to the root in a fully pipelined fashion (namely, without stopping until exhausting all the edges they know of). Finally, the root starts getting such messages at time $Diam(G)$ at the latest. □

5. The complexity of the combined algorithm. Combining the two parts, we get the following distributed algorithm for MST.

1. Perform Algorithm `Controlled-GHS` for I phases.
2. Perform Algorithm `Pipeline`.

Summarizing the results of the last two sections, we get the following theorem.

THEOREM 5.1. *There exists a distributed MST algorithm with time complexity $O(Diam(G) + n^{\varepsilon} \cdot \log^* n)$ for $\varepsilon = \frac{\ln 3}{\ln 6} = 0.6131\ldots$.*

Proof. The complexities of the two parts of our algorithm are as follows, for the given parameter I specified for the first part:

Part I: $3^I \cdot O(\log^* n)$.

Part II: $Diam(G) + \frac{n}{2^I}$.

The total time complexity is thus optimized when choosing I such that $3^I = \frac{n}{2^I}$, namely, $I = \frac{\ln n}{\ln 6}$. For this choice of I, we get $3^I = \frac{n}{2^I} = n^{\varepsilon}$ for $\varepsilon = \frac{\ln 3}{\ln 6}$, which yields a total time complexity of $O(Diam(G) + n^{\varepsilon} \cdot \log^* n)$. □

Acknowledgments. It is a pleasure to thank Ambuj Singh and Jerry James for useful comments. Thanks are also due to two anonymous referees for their comments, which helped to significantly improve the readability of the paper.

REFERENCES

[ADDJ] I. Althöfer, G. Das, D. Dobkin, and D. Joseph, *Generating sparse spanners for weighted graphs*, in Proc. 2nd Scandinavian Workshop on Algorithm Theory, Lecture Notes in Comput. Sci. 447, Springer-Verlag, New York, 1990, pp. 26–37.

[A1] B. Awerbuch, *Optimal distributed algorithms for minimum-weight spanning tree, counting, leader election and related problems*, in Proc. 19th ACM Symp. on Theory of Computing, ACM, New York, 1987, pp. 230–240.

[A2] B. Awerbuch, *Complexity of network synchronization*, J. ACM, 32 (1985), pp. 804–823.

[AGLP] B. Awerbuch, A. Goldberg, M. Luby, and S. Plotkin, *Network decomposition and locality in distributed computation*, in Proc. 30th IEEE Symp. on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1989, pp. 364–375.

[AGPV] B. Awerbuch, O. Goldreich, D. Peleg, and R. Vainish, *A tradeoff between information and communication in broadcast protocols*, J. ACM, 37 (1990), pp. 238–256.

[AKP] B. Awerbuch, S. Kutten, and D. Peleg, *Competitive distributed job load balancing*, Proc. 24th ACM Symp. on Theory of Computing, ACM, New York, 1992, pp. 571–580.

[AP1] B. Awerbuch and D. Peleg, *Network synchronization with polylogarithmic overhead*, in 31st IEEE Symp. on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 514–522.

[AP2] B. Awerbuch and D. Peleg, *Concurrent online tracking of mobile users*, Proc. ACM SIGCOMM Symposium on Communication, Architectures and Protocols, ACM, New York, 1991, pp. 221–233.

[B] B. Bollobás, *Extremal Graph Theory*, Academic Press, New York, 1978.

[CT] F. Chin and H. F. Ting, *An almost linear time and $O(n\log(n)+e)$ messages distributed algorithm for minimum-weight spanning trees*, in Proc. 26th IEEE Symp. on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1985, pp. 257–266.

[G] E. Gafni, *Improvements in the time complexity of two message-optimal election algorithms*, in Proc. 4th Symp. on Principles of Distributed Computing, ACM, New York, 1995, pp. 175–185.

[GHS] R. Gallager, P. Humblet, and P. Spira, *A distributed algorithm for minimum-weight spanning trees*, in ACM Trans. Programming Lang. Systems, 5 (1983), pp. 66–77.

[GKP] J. Garay, S. Kutten, and D. Peleg, *A sub-linear time distributed algorithm for minimum-weight spanning trees*, in Proc. 34th IEEE Symp. on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1993, pp. 659–668.

[GPS] A. V. Goldberg, S. Plotkin, and G. Shannon, *Parallel symmetry breaking in sparse graphs*, in Proc. 19th ACM Symp. on Theory of Computing, ACM, New York, 1987, pp. 315–324.

[L] N. Linial, *Distributive graph algorithms—global solutions from local data*, in Proc. 28th IEEE Symp. on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1987, pp. 331–335.

[MRR] J. M. McQuillan, I. Richer, and E. C. Rosen, *The new routing algorithm for the ARPANET*, IEEE Trans. Comm., COM-28 (1980), pp. 711–719.

[P] D. Peleg, *Time-optimal leader election in general networks*, J. Parallel Distrib. Comput., 8 (1990), pp. 96–99.

[PS1] D. Peleg and A. A. Schäffer, *Graph spanners*, J. Graph Theory, 13 (1989), pp. 99–116.

[PS2] A. Panconesi and A. Srinivasan, *Improved distributed algorithms for coloring and network decomposition problems*, in Proc. 24th ACM Symp. on Theory of Computing, ACM, New York, 1992, pp. 581–592.

[T] R. E. Tarjan, *Data Structures and Network Algorithms*, SIAM, Philadelphia, PA, 1983.

SIAM J. COMPUT.
Vol. 27, No. 1, p. 317, February 1998
© 1998 Society for Industrial and Applied Mathematics
014

ERRATUM: CONDITIONS FOR OPTIMALITY OF THE HUFFMAN ALGORITHM*

D. STOTT PARKER, JR.†

Key words. Huffman algorithm, optimal tree construction, weighted path length, tree height, quasi-linear functions, convex functions, Rényi entropy

AMS subject classifications. 94A15, 94A17, 94A24, 94A29, 94A45, 68Q20, 68R05, 05A05, 05C05

PII. S0097539797328550

The article [1] contains errors that are corrected by the underlined text below.

- page 473:
  A weight combination function $F : U^2 \to U$ is *increasing* if $\underline{y < z}$ implies $\underline{F(x, y) < F(x, z)}$ for all x, y, z in U.
- page 474:
  QL2. (Increasingness) $\underline{F(u, x) < F(u, y)}$ if $\underline{x < y}$.
- page 475:
  THEOREM 2. *F is Huffman monotone if and only if F satisfies properties* QL3 *and* QL4*, given that F is increasing.*
- page 476:
  COROLLARY 1. *F is Huffman monotone if and only if F is quasi-linear and satisfies the restrictions of Lemma* 1*, given that F is increasing.*
- page 487:
  If cost is determined by the tree's root weight, then the Hu-Tucker modification of Huffman's algorithm produces an optimal binary search tree whenever a nonshrinking weight combination function F satisfying conditions QL1, QL2, QL3, and QL4 of section 3 is used.

Acknowledgments. The importance of these errors was kindly pointed out to the author by D.E. Knuth, who constructed a very complex counterexample to Theorem 2 as it was stated in [1]. Where the article defined an increasing function F to satisfy $F(u, x) \leq F(u, y)$ if $x \leq y$, the strict definition of increasingness above is needed to establish Lemma 2 and the other results itemized. The corrections above are listed in the references of [2]. Their overdue appearance here is prompted by publication of a related work [3].

REFERENCES

[1] D. S. PARKER, *Conditions for optimality of the Huffman algorithm*, SIAM J. Comput., 9 (1980), pp. 470–489.
[2] D. E. KNUTH, *Huffman's algorithm via algebra*, J. Combin. Theory Ser. A, 32 (1982), pp. 216–224.
[3] D. S. PARKER AND P. RAM, *The construction of Huffman codes is a submodular ('convex') optimization problem over a lattice of binary trees*, SIAM J. Comput., to appear.

*Received by the editors August 21, 1997; accepted for publication October 6, 1997.
http://www.siam.org/journals/sicomp/27-1/32855.html
†UCLA Computer Science Department, University of California, Los Angeles, CA 90095-1596 (stott@cs.ucla.edu).

SIAM J. COMPUT.
Vol. 27, No. 2, pp. 319–340, April 1998
© 1998 Society for Industrial and Applied Mathematics

ATOMIC SNAPSHOTS IN $O(n \log n)$ OPERATIONS*

HAGIT ATTIYA† AND OPHIR RACHMAN†

Abstract. The *atomic snapshot object* is an important primitive used for the design and verification of wait-free algorithms in shared-memory distributed systems. A snapshot object is a shared data structure partitioned into segments. Processors can either *update* an individual segment or instantaneously *scan* all segments of the object. This paper presents an implementation of an atomic snapshot object in which each high-level operation (scan or update) requires $O(n \log n)$ low-level operations on atomic read/write registers.

Key words. atomic read/write registers, single-reader multiwriter, snapshot objects, linearizability, asynchronous shared memory systems, wait-free computations

AMS subject classifications. 68P05, 68Q10, 68Q20, 68Q22

PII. S0097539795279463

1. Introduction.

Wait-free algorithms for shared-memory systems have attracted considerable attention during the past few years. The difficulty of synchronization and communication in such systems caused many of the algorithms that were developed to be quite intricate. A major research effort attempts to simplify the design and verification of efficient wait-free algorithms by defining convenient synchronization primitives and efficiently implementing them. One of the most attractive primitives is the *atomic snapshot object* introduced in [1, 2, 6].

An atomic snapshot object (in short, *snapshot object*) is a data structure shared by n processors. The snapshot object is partitioned into n segments, one for each processor. Processors can either *update* their own segment or instantaneously *scan* all segments of the object. By employing a snapshot object, processors obtain an instantaneous global picture of the system. This sidesteps the need to rely on "inconsistent" views of the shared memory and reduces the possible interleavings of the low level operations in the execution. Therefore, snapshot objects greatly simplify the design and verification of many wait-free algorithms. An excellent example is provided by comparing the recent proof of a bounded concurrent timestamp algorithm using snapshot objects [15] with the original intricate proof in [10].

Unfortunately, the great conceptual gain of using snapshot objects is often diminished by the actual cost of their implementation; the best snapshot implementation to date requires $O(n^2)$ read and write operations on atomic registers [1, 4]. Compared with the cost of simply reading n memory locations, this might seem a high price to pay for modularity and transparency. Thus, significant effort has been spent on avoiding snapshots and constructing algorithms directly from read and write operations.

This paper presents a snapshot object implementation in which each update or scan operation requires $O(n \log n)$ operations on single-writer multireader atomic reg-

*Received by the editors January 3, 1995; accepted for publication (in revised form) December 18, 1995. An extended abstract of this paper appeared in *Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing*, Association for Computing Machinery, New York, 1993, pp. 29–40. This research was supported by grant 92-0233 from the United States–Israel Binational Science Foundation (BSF), Jerusalem, Israel, by the Technion V.P.R., Argentinian Research Fund, and by the fund for the promotion of research in the Technion.

http://www.siam.org/journals/sicomp/27-2/27946.html

†Department of Computer Science, The Technion, Haifa 32000, Israel (hagit@cs.technion.ac.il, fimfam@cs.technion.ac.il).

isters. Thus, we dramatically reduce the gap between the trivial lower bound of $\Omega(n)$ and the best known upper bound of $O(n^2)$ for atomic snapshots. Consequently, our snapshot object makes it feasible to design modular and easy to verify wait-free algorithms, without a great sacrifice in their efficiency.

We start with an algorithm for implementing an m-shot snapshot object, that is, a snapshot object to which up to m operations can be applied. The algorithm is simple and requires $O(n \log m)$ operations on single-writer multireader atomic registers. The algorithm is inspired by the algorithm presented in [7] for solving *lattice agreement* [4, 7, 11]. However, the algorithm of [7] uses atomic Test&Set operations, while the current algorithm uses only atomic read and write operations.

We then present ways to transform this algorithm to implement an ∞-shot snapshot object, that is, an object that supports an infinite number of operations.

One way is based on general-purpose transformations. In [7], the snapshot object was proved to be reducible to the lattice agreement problem. By employing the transformation of [7], the restriction of our algorithm to solve lattice agreement immediately implies an ∞-shot snapshot object in which each operation requires $O(n \log n)$ read and write operations on atomic registers. Unfortunately, this implementation requires an unbounded amount of memory. The bounded rounds abstraction of [13] can be used to bound the memory requirements of this implementation.

An alternative path is a direct implementation of an ∞-shot snapshot object, with $O(n \log n)$ operations for each scan or update. This implementation uses a bounded amount of memory and is based on recycling a single copy of the m-shot object. This recycling combines in a novel way synchronization techniques such as handshake bits [6], borrowing views [1] and traceable use techniques [14], and we believe it is interesting on its own.

The bounded algorithm uses atomic operations on registers that may contain up to $O(n(\log n + |V|))$ bits, where $|V|$ is the number of bits needed to represent a value of the snapshot object. (There are also operations on registers of size $O(n^4 \log n)$, but these occur infrequently.)

Besides the conceptual contribution to the design of future wait-free algorithms, our snapshot object immediately yields improvements to existing algorithms that use the snapshot object by plugging in our more efficient one. These include randomized consensus [3, 6], approximate agreement [8], bounded timestamping [15], and general constructions of wait-free concurrent objects [4, 17].

A *multiwriter* snapshot object is a generalized snapshot object in which any processor can update any segment. There is a transformation of Anderson's [2] which uses any snapshot object as a black box to construct a multiwriter snapshot object; this transformation requires a linear number of read and write operations. This transformation can be used to turn our algorithm into an algorithm for a multiwriter snapshot object with the same complexity.

Deterministic snapshot implementations have been proposed by Anderson [2] (bounded memory and exponential number of operations), by Aspnes and Herlihy [4] (unbounded memory and $O(n^2)$ operations), and by Afek et al. [1] (bounded memory and $O(n^2)$ operations). Attiya, Herlihy, and Rachman [7] give an $O(n \log^2 n)$ implementation that uses *Test&Set* registers, and an $O(n)$ implementation that uses dynamic *Test&Set* registers. Israeli, Shaham, and Shirazi [23] give a general technique to transform any snapshot implementation that requires $O(f(n))$ operations per scan or update into an (unbounded) implementation that requires $O(f(n))$ operations per scan and only a linear number of operations per update (or vice versa). Constructions of multiwriter snapshot objects appear in [1, 2].

Introduced in [7] are randomized implementations of the snapshot object ($O(n \log^2 n)$ using single-writer multireader registers, and $O(n)$ using dynamic single-writer multireader registers). Chandra and Dwork [9] also give a randomized implementation that requires $O(n \log^2 n)$ operations on atomic single-writer multireader registers. Weaker variants of the snapshot object were implemented by Kirousis, Spirakis, and Tsigas [24] (single-scanner snapshot object), and by Dwork et al. [12] (nonlinearizable snapshot object).

Independent of our work, Israeli and Shirazi [21] constructed a deterministic snapshot object that requires $O(n^{3/2} \log^2 n)$ operations and uses unbounded memory. Also, they showed a lower bound of $\Omega(\min\{w, r\})$ low-level operations for any update operation, where w is the number of updaters and r is the number of scanners [22].

As is made clear by the above review, our $O(n \log n)$ deterministic snapshot implementation significantly improves all known deterministic implementations that use only atomic registers and even improves almost all the existing randomized implementations. Note that by the general technique of [23], our snapshot implementation can be improved to require $O(n \log n)$ operations per update and only $O(n)$ operations per scan (or vice versa).

Following the original publication of our algorithm, Inoue et al. [19] presented an atomic snapshot object that requires only a linear number of read and write operations. However, this algorithm requires multiwriter registers, that is, each processor can write to each register. In contrast, our algorithms use only single-writer registers.

The rest of the paper is organized as follows. Section 2 includes definitions of the model and of the snapshot object. In section 3, we present the implementation of the m-shot snapshot object, which is then used to construct the ∞-shot snapshot object in section 4. We conclude in section 5 with a discussion of our results.

2. The snapshot object. Our model of computation is standard and follows, e.g., [8, 16].

An *atomic snapshot* object is partitioned into n segments, $S_1, \ldots, S_n$, where only processor p_i may write to the ith segment. The snapshot object supports two operations, *scan* and *update*(v). The scan operation allows a processor to obtain an instantaneous view of the segments, as if all n segments are read in a single atomic step. A scan operation returns a *view*, which is a vector $V[1, \ldots, n]$, where $V[i]$ is the value for the ith segment. The update operation with parameter v allows a processor to write the value v into its segment.

An implementation of the snapshot object should be *linearizable* [18]. That is, any execution of *scan* and *update* operations should appear as if it was executed sequentially in some order that preserves the real time order of the operations.

In more detail, each scan or update is implemented as a sequence of primitive operations. The nature of the primitive operations depends on the low-level objects used; in our case, read and write operations of atomic registers. An execution is a sequence of primitive operations, each executed by some processor as part of some scan or update operation. We assume that each processor has at most one (high-level) operation in progress at a time; that is, it does not start a new operation before the preceding one has completed.

Define a partial order $\rightarrow$ on (high-level) operations in an execution such that $op_1 \rightarrow op_2$ if (and only if) the operation op_1 has terminated before the operation op_2 has started; that is, all low-level operations that comprise op_1 appear in the execution before any low-level operation that is part of op_2. The partial order $\rightarrow$ reflects the external real time order of nonoverlapping operations in the execution.

For the snapshot implementation to be correct, we require that *scan* and *update* operations can be *linearized*. That is, there is a sequence that contains all *scan* and *update* operations in the execution that

a. extends the real time order of operations as defined by the partial order $\rightarrow$; and
b. obeys the sequential semantics of the snapshot operations; that is, if *view* is returned by some *scan* operation, then for every segment i, $view[i]$ is the value written by the last update to the ith segment which precedes the scan operation in the sequence.

In this paper, we define one operation that combines both scan and update, denoted $scate(v)$. Executing a $scate(v)$ operation by p_i both writes v into S_i and returns an instantaneous view of the n segments.[1] Intuitively, to perform $update(v)$ a processor invokes $scate(v)$ and simply ignores the view it returns; to perform a scan the processor invokes $scate(v)$, where v is the current value of its segment.

Another property that we require is *wait-freedom*; that is, every processor completes its execution of a scan or an update within a bounded number of its own (low-level) operations, regardless of the execution of other processors.

3. Implementation of an m-shot snapshot object. In this section we construct an m-shot snapshot object, which is a degenerate instance of the general snapshot object. Namely, an m-shot snapshot object is defined exactly as the general snapshot object, except that the total number of scate operations that may be performed by all processors is at most m.

3.1. Preliminaries. For the construction of the m-shot snapshot object, we modify each segment of the snapshot object to contain both the value of the segment in the field *value*, and some additional information that indicates the number of times p_i performed an operation. The additional fields *seq* and *counter* are incremented with each operation performed by p_i. Although the *seq* and *counter* fields contain exactly the same information, they have different roles in the implementation. The *seq* field determines which of two values written by p_i is more up to date. The *counter* field simply counts the number of operations performed by p_i. When we present the general implementation of the snapshot object, we shall see that the information in these two fields is maintained differently; this is why we separate them here as well.

Note that each *scate* operation returns a *view*, which is a vector with three fields in each entry. All segments are initially $(\perp, 0, 0)$. We now introduce some terminology.

The *size* of a view V, denoted by $|V|$, is $\sum_i V[i].counter$. The size of a view reflects the "amount of knowledge" that this view contains; that is, the size of a view counts the total number of operations performed on the snapshot object before this view was obtained.

A view V_1 *dominates* a view V_2, if for all i, $V_1[i].seq \geq V_2[i].seq$. Two views V_1 and V_2 are *comparable* if either V_1 dominates V_2 or V_2 dominates V_1. Two views V_1 and V_2 are *unanimous*, if for all i, $V_1[i].seq = V_2[i].seq$ implies that $V_1[i] = V_2[i]$. A set $\{V_1, \ldots, V_l\}$ of views is *unanimous* if any pair of views in the set are unanimous. The *union* of a unanimous set $\{V_1, \ldots, V_l\}$ of views, denoted by $\cup\{V_1, \ldots, V_l\}$, is the minimal view that dominates all views $V_1, \ldots, V_l$. That is, the union is a view V_u such that for every i, $V_u[i]$ equals $V_j[i]$ with maximal *seq* field. (All the sets of views that we use in the paper are trivially unanimous. Therefore we use unions of sets of views without explicitly stating that the sets are unanimous.)

[1] Combining the roles of scans and updates was implicitly done in previous works, where update operations not only write new values but also return views.

Classifier(K, I_i): returns(O_i) (Code for p_i)
0: write I_i to R_i
1: read $R_1, \ldots, R_n$
2: **if** $|\cup\{R_1, \ldots, R_n\}| > K$ **then**
3: read $R_1, \ldots, R_n$ and **return**$(O_i = \cup\{R_1, \ldots, R_n\})$
4: **else return**$(O_i = I_i)$

FIG. 1. *The classifier procedure.*

3.2. The *classifier* procedure. We start by introducing a procedure called *classifier*, with parameter K. Each processor p_i starts the procedure with an input view I_i, and upon termination, returns an output view O_i. The *classifier* procedure appears in Figure 1. The processors use a set of single-writer multireader registers $R_1, \ldots, R_n$.

In the *classifier* procedure each processor p_i starts with some local knowledge that is held in I_i. The goal of the *classifier* procedure is to update the processors' knowledge in some organized manner. Roughly speaking, the processors that use the procedure are classified into two groups such that the processors in the first group retain their original knowledge, while each processor in the second group increases its knowledge to dominate the knowledge of all the processors in the first group. Specifically, processors in the first group are called *slaves* and are defined as the processors that terminate the procedure in line 4. Processors in the second group are called *masters* and are defined as the processors that terminate the procedure in line 3. The classifying property of the procedure is the crux of the m-shot snapshot object. Notice that the *classifier* procedure provides very little guarantee on the number of masters and slaves. In particular, it is possible that all processors are classified as masters.

3.3. The implementation. To implement the scate operation for an m-shot snapshot object, we construct a full binary tree with $\log m$ levels and $m-1$ nodes.[2] The nodes of the tree are labeled by an in-order numbering on the tree, assigning labels in increasing order from the set $\{1, \ldots, m-1\}$. For each node v, we denote the label of v by $Label(v)$. The labels given by the in-order search can be presented in the following recursive manner: the root (in level 1) is labeled $\frac{m}{2}$; inductively, if a node v in level ℓ is labeled $Label(v)$, then the left child of v, denoted by $v.left$, is labeled $Label(v) - \frac{m}{2^{\ell+1}}$, and the right child of v, denoted by $v.right$, is labeled $Label(v) + \frac{m}{2^{\ell+1}}$. (See Figure 2.)

Since each processor may perform several scate operations, we do not identify an operation with the processor that executes it. In the rest of the section, we refer to operations as independent entities that "execute themselves."

Intuitively, each operation traverses the tree downwards starting from the root. Inside the tree, operations that arrive at some node execute the *classifier* procedure using the label of the node as the parameter K. The *classifier* procedure of each node separates the arriving operations so that less knowledgeable operations (slaves) proceed to the left and more knowledgeable operations (masters) proceed to the right. This process continues throughout the levels of the tree; an operation terminates when it arrives at a leaf of the tree.

The main idea in this construction is that operations are ordered in the leaves (from left to right) according to the amount of knowledge they have collected. As we prove in the following, when two operations are separated by some node, then the final

[2] We assume m is an integral power of 2. Otherwise, standard padding techniques can be applied.

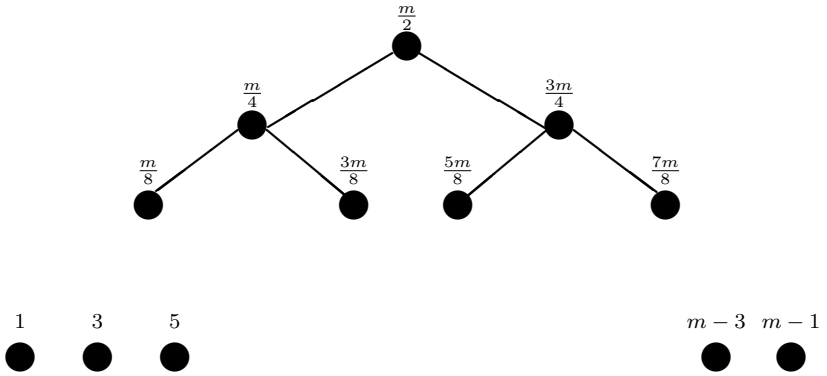


FIG. 2. *The classification tree.*

knowledge of the operation that proceeded to the right dominates the final knowledge of the operation that proceeded to the left. This guarantees that operations arriving at different leaves are comparable and are ordered from left to right. In addition, we prove that if two operations traverse exactly the same path in the tree, then they have exactly the same final knowledge. Such two operations undergo a "squeezing" process, where the difference in their knowledge is constantly reduced as they move toward the leaves of the tree. Finally, when two operations arrive at the same leaf the difference in their knowledge is squeezed to zero, and they are forced to have exactly the same knowledge.

To implement this intuitive idea, we associate with each node a separate area in the shared memory that contains a set of n single-writer multireader registers $R_1, \ldots, R_n$. These registers are initialized to empty views that contain $(\perp, 0, 0)$ in each entry and are used to execute the *classifier* procedure at that node. In addition, each processor p_i has a local variable $current_i$ that is initialized at the beginning of each operation by p_i. This local variable stores the accumulated knowledge of p_i during the execution of the operation. For ease of exposition, we add a $(\log m + 1)$th level to the tree, which now contains the leaves of the tree. These leaves have no labels and no associated registers, and they serve only as "final stations" for the operations that traverse the tree.

All *scate* operations, up to m, are executed on the tree constructed above. A *scate* operation *op* by p_i is executed as follows: first, *op* writes the value of the operation into S_i. Second, *op* reads the n segments $S_1, \ldots, S_n$, and sets $current_i$ to hold the view that contains the values read from the segments. Then *op* starts traversing the tree by entering the root. In general, when *op* enters some node v, it uses the value of $current_i$ as an input vector I_i, executes a $classifier(Label(v), current_i)$ procedure at v, and updates its $current_i$ variable to hold the value returned by the procedure. If p_i terminates the *classifier* procedure in v as a master, it enters $v.right$; otherwise it enters $v.left$. When *op* enters a leaf, it terminates and returns the value of $current_i$ as its final view. For clarity, we denote by $current_{i,\ell}$ the value of $current_i$ as *op* enters level ℓ. The precise code for a scate operation (by p_i) appears in Figure 3.

3.4. Correctness proof. We start by stating the properties of the *classifier* procedures that are executed in the various nodes of the tree. We first introduce some notation. For each node v, $Ops(v)$ denotes the set of operations that traverse through v. At each node v, each operation in $Ops(v)$ is classified either as a master or as a slave. The set of operations that are classified as masters at v is denoted by

scate(*value*) (Code for p_i)
1: $S_i := (value, incremented\ seq\ and\ counter)$
2: **for** $j := 1, \ldots, n$ $current_{i,1}[j] := S_j$
3: $v := root$
4: **for** $\ell := 1, \ldots, \log m$ **do**
5: $current_{i,\ell+1} :=$*Classifier*$(Label(v), current_{i,\ell})$
6: **if** master **then** *v:=v.right*
7: **if** slave **then** *v:=v.left*
8: **return**$(current_{i,\log m+1})$

FIG. 3. *The code for a scate operation.*

$M(v)$, and the set of operations that are classified as slaves at v is denoted by $S(v)$. In addition, we denote the input view of an operation op_i for the *classifier* procedure at level ℓ by $I_{i,\ell}$. (If p_j is the processor that executes op_i, then $I_{i,\ell}$ is the value assigned to $current_{j,\ell}$ during op_i.)

LEMMA 3.1. *Let v be some node at level ℓ. Let L and H be nonnegative integers such that $L \leq Label(v) \leq H$. If $L < |I_{i,\ell}| \leq H$ for every $op_i \in Ops(v)$, and $|\cup\{I_{i,\ell} : op_i \in Ops(v)\}| \leq H$, then*
(b1) *for every $op_i \in M(v)$, $Label(v) < |I_{i,\ell+1}| \leq H$,*
(b2) *for every $op_i \in S(v)$, $L < |I_{i,\ell+1}| \leq Label(v)$,*
(b3) $|\cup\{I_{i,\ell+1} : op_i \in M(v)\}| \leq H$,
(b4) $|\cup\{I_{i,\ell+1} : op_i \in S(v)\}| \leq Label(v)$, *and*
(b5) *for every $op_i \in M(v)$, $I_{i,\ell+1}$ dominates $\cup\{I_{j,\ell+1} : op_j \in S(v)\}$.*

Proof. Properties (b1)–(b3) are immediate from the code.

Property (b4) is proved by contradiction. Assume that $|\cup\{I_{i,\ell+1} : op_i \in S(v)\}| > Label(v)$. Since for each $op_i \in S(v)$ we have $I_{i,\ell+1} = I_{i,\ell}$, it follows that $|\cup\{I_{i,\ell} : op_i \in S(v)\}| > Label(v)$. Let op_j be the last operation in $S(v)$ that executes line 0 in the *classifier* procedure of v. When op_j executes line 1 of the procedure, all $I_{i,\ell}$ such that $op_i \in S(v)$ are already written in the registers of v. Since the value in any register of any node is overwritten only with values that dominate it, op_j collects a view with size greater than $Label(v)$. This contradicts the assumption that $op_j \in S(v)$.

To prove (b5), we show that when $op_i \in M(v)$ starts to execute line 3 in the *classifier* procedure of v, all $\{I_{j,\ell} : op_j \in S(v)\}$ are already written in the registers of v. Otherwise, if some $op_j \in S(v)$ has not yet written $I_{j,\ell}$, then when op_j executes line 1 of the procedure it reads registers' values that dominate the registers' values that op_i read in line 1. This contradicts the assumption that $op_j \in S(v)$. □

Using the properties of the *classifier* procedure as stated in the above lemma, we now prove that all the views returned in scate operations are comparable. To show that, we first prove that the views returned by operations that terminate in different leaves of the tree are comparable. The following two simple lemmas are implied by the code.

LEMMA 3.2. *Let op_i be an operation that returns V_i. Let v be a node such that $op_i \in S(v)$ and let ℓ be v's level. Then V_i is dominated by $\cup\{I_{j,\ell+1} : op_j \in S(v)\}$.*

LEMMA 3.3. *Let op_i be an operation that returns V_i. Let v be a node such that $op_i \in M(v)$ and let ℓ be v's level. Then V_i dominates $I_{j,\ell+1}$ for any $op_j \in S(v)$.*

The next lemma uses Lemmas 3.2 and 3.3 to prove that the views returned by operations that terminate in different leaves are comparable.

LEMMA 3.4. *Let op_i and op_j be two operations that terminate in leaves v_i and v_j, respectively, where $v_i \neq v_j$. Then the views returned by op_i and op_j are comparable.*

Proof. Let v be the node with maximal level (closest to the leaves) such that both op_i and op_j belong to $Ops(v)$, and let ℓ be its level. Since $v_i \neq v_j$, $\ell < \log m + 1$, that is, v is an inner node. Since v is not a leaf, one of op_i and op_j is a master in v, and the other is a slave at v. Assume, without loss of generality, that $op_i \in S(v)$ and $op_j \in M(v)$.

By Lemma 3.2, the view returned by op_i is dominated by $\cup\{I_{k,\ell+1} : op_k \in S(v)\}$. By Lemma 3.3, the view returned by op_j dominates each $I_{k,\ell+1}$, if $op_k \in S(v)$, and therefore dominates $\cup\{I_{k,\ell+1}, op_k \in S(v)\}$. Thus, the view returned by op_j dominates the view returned by op_i. □

To complete the comparability proof, we show that the output views of operations that terminate in the same leaf are comparable. The next lemma formally captures the intuitive idea of the "squeezed" difference in knowledge. The lemma bounds the size of the inputs $I_{i,\ell}$ and their union at some node v of some level ℓ as a function of $Label(v)$ and ℓ.

LEMMA 3.5. *Let v be an inner node of level ℓ. Then,*

(1) *for every $op_i \in Ops(v)$, $Label(v) - \frac{m}{2^\ell} < |I_{i,\ell}| \leq Label(v) + \frac{m}{2^\ell}$, and*
(2) $|\cup\{I_{i,\ell} : op_i \in Ops(v)\}| \leq Label(v) + \frac{m}{2^\ell}$.

Proof. The proof is by induction on ℓ. For the induction base $\ell = 1$, the lemma is straightforward since the total number of operations is at most m. For the induction step, assume the lemma holds for all nodes in level $\ell - 1$ and consider an arbitrary node v in level $\ell > 1$. Let v' be the parent of v and consider the *classifier* procedure with parameter $K = Label(v')$ that is executed by $Ops(v')$ in v'. By the induction hypothesis we have

(1) $Label(v') - \frac{m}{2^{\ell-1}} < |I_{i,\ell-1}| \leq Label(v') + \frac{m}{2^{\ell-1}}$, for any $op_i \in Ops(v')$, and
(2) $|\cup\{I_{i,\ell-1} : op_i \in Ops(v')\}| \leq Label(v') + \frac{m}{2^{\ell-1}}$.

If we denote $L = Label(v') - \frac{m}{2^{\ell-1}}$ and $H = Label(v') + \frac{m}{2^{\ell-1}}$, then these are exactly the conditions of Lemma 3.1. We have two cases.

Case 1. If $v = v'.right$, then $K = Label(v') = Label(v) - \frac{m}{2^\ell}$, and $Ops(v) = M(v')$. We have by (b1) and (b3) of Lemma 3.1 that

(1) for any $op_i \in Ops(v)$, $Label(v) - \frac{m}{2^\ell} < |I_{i,\ell}| \leq Label(v) + \frac{m}{2^\ell}$, and
(2) $|\cup\{I_{i,\ell} : op_i \in Ops(v)\}| \leq Label(v) + \frac{m}{2^\ell}$,

which are the required conditions for the operations in $Ops(v)$.

Case 2. If $v = v'.left$, then $K = Label(v') = Label(v) + \frac{m}{2^\ell}$, and $Ops(v) = S(v')$. In this case, the same equations are implied by (b2) and (b4) of Lemma 3.1. □

The next lemma proves that the views returned by two operations that terminate at the same leaf are equal, and in particular, comparable.

LEMMA 3.6. *Let op_i and op_j be two operations that terminate in the same leaf v. Then the views returned by op_i and op_j are equal.*

Proof. Let v' be the parent of v. Assume, without loss of generality, that $v = v'.right$; the proof if v is the left child of v' follows in the same manner. By Lemma 3.5, since v' is in level $\ell = \log m$,

(1) $Label(v') - 1 < |I_{k,\log m}| \leq Label(v') + 1$, for any $op_k \in Ops(v')$, and
(2) $|\cup\{I_{k,\log m} : op_k \in Ops(v')\}| \leq Label(v') + 1$.

The operations op_i and op_j execute the *classifier* procedure in v' with parameter $K = Label(v')$ and both terminate as masters and proceed to v. If we denote $L = Label(v') - 1$, and $H = Label(v') + 1$, then conditions (1) and (2) above are the required conditions for applying Lemma 3.1 to the *classifier* procedure that is executed in v'. Thus, by Lemma 3.1(b1), since op_i and op_j are in $M(v')$, we have

$|I_{i,\log m+1}| = |I_{j,\log m+1}| = \mathit{Label}(v') + 1$. In addition, by Lemma 3.1(b3), we have $|\cup \{I_{i,\log m+1}, I_{j,\log m+1}\}| = \mathit{Label}(v') + 1$. Therefore, $I_{i,\log m+1} = I_{j,\log m+1}$, which implies that the output views of op_i and op_j are equal. □

Lemmas 3.4 and 3.6 prove that the views returned by the scate operations are comparable. We now use these comparable scate operations to implement the linearizable scan and update operations of the snapshot object.

To execute an update(v) operation, a processor simply executes a scate(v) operation and ignores the view it returns. To execute a scan operation, a processor executes a scate(v) operation using the current value of its segment. Notice that although the same value is used, the *seq* and *counter* values are incremented. Thus, a scan operation by p_i changes the sequence number of S_i but does not change the value of S_i. Also notice that both scan and update operations return views. These views are later used for the linearization of the scan and update operations.

In order to define the linearization of operations on the snapshot object, we first order the scan operations and then insert the update operations. Consider any two scan operations sc_i and sc_j that return V_i and V_j, respectively. If $V_i \neq V_j$ and V_j dominates V_i, then sc_i is linearized before sc_j and vice versa if V_i dominates V_j. If $V_i = V_j$, then we order them first by the partial order $\rightarrow$, and if the operations are not ordered with respect to $\rightarrow$, then we break symmetry by the identities of the processors that execute the operations. This ordering of scans is well defined since a processor has only one operation outstanding at a time, and hence two operations by the same processor are always ordered by $\rightarrow$.

Next, we insert the update operations between the linearized scan operations. Consider an update operation that wrote a value (v, seq) to some segment S_i. The update operation is linearized after the last scan operation that returns a view that does not contain (v, seq) and before the first scan operation that contains (v, seq). Since scan operations are ordered by their views, each update operation fits exactly between two successive scan operations. We break symmetry between update operations that fit between the same two scan operations in the same manner as in the scan operations, that is, first by the partial order $\rightarrow$ and then by processors' identities. We now prove that this sequence is a linearization.

The next lemma follows immediately from the way update operations are linearized between scan operations.

LEMMA 3.7. *For any scan operation sc and for all segments S_i, the value returned by sc for S_i is the value written by the last update operation by processor p_i that is linearized before sc.*

Therefore, the linearization sequence we constructed preserves the semantics of the snapshot object. We now prove that it extends the partial order $\rightarrow$.

LEMMA 3.8. *For any two (scan/update) operations op_i and op_j, if $op_i \rightarrow op_j$ then op_i is linearized before op_j.*

Proof. There are four cases, according to operation types.

Case 1. Let sc_i and sc_j be two scan operations such that $sc_i \rightarrow sc_j$. By the code of the algorithm, the view returned by sc_i does not dominate the view returned by sc_j and hence the view returned by sc_j dominates the view returned by sc_i. Since scan operations are linearized by their views, this implies that sc_i is linearized before sc_j.

Case 2. Let sc_i be a scan operation and up_j be an update operation such that $sc_i \rightarrow up_j$. By the code of the algorithm, the view returned by sc_i does not contain the value written by up_j, and therefore, up_j is linearized after sc_i.

Case 3. Let up_i be an update operation and sc_j be a scan operation such that $up_i \rightarrow sc_j$. By the code of the algorithm, the view returned by sc_j contains the value written by up_i (or a value written by a later update operation by p_i,) and therefore, up_i is linearized before sc_j.

Case 4. Let up_i and up_j be two update operations such that $up_i \rightarrow up_j$. If up_i and up_j fit exactly between the same two scan operations, then due to the way symmetry is broken up_i is linearized before up_j, and the claim follows.

Otherwise, assume by way of contradiction that there exists a scan operation sc such that up_j is linearized before sc and sc is linearized before up_i. Thus, sc returns a view that contains the value written by up_j and does not contain the value written by up_i. Consider the scate operation that is executed to implement up_i. This scate operation returns a view that contains the value written by up_i but does not contain the value written by up_j. Therefore, this scate operation returns a view that is incomparable to the view returned by sc. This contradicts the comparability property of the views returned by the scate operations (Lemmas 3.4 and 3.6). □

Lemmas 3.7 and 3.8 prove that the scate operation of Figure 3 implements an m-shot snapshot object. The complexity analysis is obvious, and we have the following theorem.

THEOREM 3.9. *Each operation on the m-shot snapshot object implemented by the scate operation of Figure* 3 *requires $O(n \log m)$ operations on atomic single-writer multireader registers.*

Note that each processor has a view for each level of the classification tree. Denote by B the number of bits required to represent a view. Since the tree has $O(m)$ nodes, and for each node we have a view for each processor, it follows that the algorithm requires a total of $O(mnB)$ bits.

4. A general snapshot object.

4.1. An unbounded algorithm. A straightforward way to transform the m-shot snapshot object into an ∞-shot one is via the lattice agreement decision problem [4, 7, 11]. In this problem, processors start with inputs from a complete lattice and have to decide (in a nontrivial manner) on comparable outputs (in the lattice). It is fairly simple to use an n-shot snapshot object to solve lattice agreement and there is a general transformation that converts any lattice agreement algorithm into an implementation of an ∞-shot snapshot object [7]. The overhead of this transformation is $O(n)$ reads and writes per scan or update operation. Therefore, the m-shot snapshot object of the previous section can be converted into an ∞-shot snapshot object in which a scan or update operation requires $O(n \log n)$ operations.

Unfortunately, the general transformation of [7] extensively uses unbounded memory. That is, the transformation (possibly) replicates the memory area required for one lattice agreement algorithm, for each operation on the snapshot object. This is a consequence of the generality of the transformation, which does not assume anything about the lattice agreement algorithm. In tailoring the transformation to our m-shot snapshot object, the memory requirements can be reduced. That is, the number of registers can be bounded, and only their values increase by one with each new operation of the snapshot object. (The details, which are straightforward, will not be discussed here.) While these memory requirements are sufficient for any practical purpose, it is theoretically interesting to construct an ∞-shot snapshot object that requires only a bounded amount of shared memory.

A method to bound the memory requirements of the general transformation appears in [13]. Here we show a direct approach for combining the ideas of the m-shot

snapshot object with synchronization mechanisms to obtain a bounded implementation of a general snapshot object.

4.2. Bounded ∞-shot snapshot object. As mentioned before, the transformation of [7] employs an infinite number of copies of a lattice agreement algorithm so that each processor executes at most one operation on each copy. The algorithm presented here uses similar ideas but with a single copy of the m-shot snapshot object of the previous section.

Recall that in the construction of the m-shot snapshot object, each segment S_i has two additional fields, *seq* and *counter*. The *counter* field indicates how many operations were performed by p_i, while *seq* distinguishes, for any two values of p_i, which is more up to date. For the bounded implementation, we maintain this information using only bounded memory. Intuitively, the *seq* field is maintained using bounded sequential timestamps; the details are discussed in section 4.4. The more difficult task is to maintain the *counter* field, used for the classification process, using bounded memory.

In the general algorithm, we use the same tree of height $\log m+1$ which is traversed by the operations, as in the m-shot object. In order to allow one tree to support an unbounded number of operations, instead of only m, the operations are divided into *virtual* rounds, each containing exactly m operations.

By appropriate control mechanisms, we separate operations from different rounds so that they are not interleaved. In this way, the behavior of operations of the same round correspond to executing m operations on a separate m-shot snapshot object.

4.2.1. The bounded counter mechanism. In the m-shot object, the *counter* field associated with each segment specifies how many times the segment was updated; summing the *counter* fields over all segments yields the total number of operations that were performed on the snapshot object. In the general implementation, the *counter* field associated with a segment specifies the number of times the segment was updated modulo m; in this way, summing the *counter* yields the total number of operations that were performed on the object modulo m. (Although this sum is actually in the range $1, \ldots, nm$, we only refer to its value modulo m.)

We use the following terminology. The *counter* fields are called the *local counters*. The sum of the local counters modulo m is the *global counter*. The values of the local counters, as well as the global counter, are in the range $0, \ldots, m-1$.

For the sake of the proof, it is convenient to consider the unbounded values of these counters as well. That is, with each local counter we associate a virtual counter with the real (unbounded) value of that counter. Summing the virtual counters defines the real value of the global counter. The real values of the counters are not used within the code but only for the analysis.

4.2.2. The handshake mechanism. In the algorithm, we need to know the chronological order of operations by different processors. Specifically, for any two processors, p_i and p_j, we wish to know how many operations p_i started since a certain point in p_j's last operation (and vice versa). Clearly, we cannot maintain the exact number of operations since it is inherently unbounded. Therefore, we only want to know if the number of operations that p_i started is either 0, 1, 2,..., $k-1$, or strictly more than $k-1$ (for some constant k). This is done with a handshake mechanism that was introduced in [6].

For every two processors, p_i and p_j, there are two handshake variables $H_{i,j}$ and $H_{j,i}$. $H_{i,j}$ is written by p_i and read by p_j, while $H_{j,i}$ is written by p_j and read by p_i. An intuitive way to describe the functionality of the handshake variables is to consider

Handshake$_i(j)$
0: $temp = H_{j,i}$
1: **if** $Dist(H_{i,j}, temp) = 0$ **then return**$(H_{i,j} + 1)$
2: **if** $Dist(H_{i,j}, temp) \leq k$ **then return**$(temp)$
3: **if** $Dist(H_{i,j}, temp) > 2k$ **then return**$(H_{i,j} + 1)$

FIG. 4. *The handshake*$_i(j)$ *procedure.*

Takeover$_i(j)$ Invoked with every read from p_j's variable
1: **if** $Dist(H_{i,j}, H_{j,i}) = k$ **then goto** Takeover by p_j code

FIG. 5. *The takeover*$_i(j)$ *procedure.*

a directed cycle with vertices numbered $0, \ldots, 3k$, where the direction is defined from t to $(t + 1)$ mod $(3k + 1)$. The variables $H_{i,j}$ and $H_{j,i}$ represent the positions of p_i and p_j on this cycle. To handshake with p_i, p_j checks the values of $H_{i,j}$ and $H_{j,i}$ and updates its own position on the cycle accordingly.

More precisely, the function *handshake*$_i(j)$ is called by p_i in order to update $H_{i,j}$ (Figure 4). Using the procedures *handshake*$_i(j)$ and *handshake*$_j(i)$ by p_i and p_j, respectively, maintains the invariant that the directed distance from $H_{i,j}$ to $H_{j,i}$ on the cycle, denoted $Dist(H_{i,j}, H_{j,i})$, is either in the range $[0, \ldots, k]$ or in the range $[2k, \ldots, 3k]$. This invariant is used to determine who is the more advanced of the two processors. If the distance from $H_{i,j}$ to $H_{j,i}$ is at most k (but not zero), then p_j is more advanced, and if the distance is between $2k$ and $3k$ then p_i is more advanced. (If the distance is zero then p_i and p_j are equally advanced.)

4.2.3. The failure detection mechanisms. In the implementation we present, a scate operation may temporarily *fail* in one of two ways. The first kind of failure occurs if some processor, say p_j, performs several operations while p_i traverses the classification tree. This kind of failure is called a *takeover* failure; when it occurs, we say that p_i was *overtaken* by p_j. The second kind of failure is a *wraparound* of the global counter, which occurs when the value of the global counter goes from m to 0 while p_i traverses the classification tree. We now describe the failures in more detail and explain the failure detection mechanisms we employ.

Takeover failures are detected by a mechanism that is constantly being operated (see Figure 5). Whenever a processor p_i reads a register of some other processor, say p_j, it checks the value of $H_{j,i}$ with respect to $H_{i,j}$. If $Dist(H_{i,j}, H_{j,i}) = k$, that is, p_j executed k or more handshakes since p_i executed its last handshake, then a takeover failure by p_j is detected. In this case, p_i jumps directly to a place in the code that handles this situation.

Wraparound failures are detected by a different mechanism. Before p_i traverses the tree, it collects the values of the local counters and computes a value for the global counter. Later, p_i checks for a wraparound by using the procedure *check-wraparound*. The procedure receives the global counter's value that p_i computed earlier and reads the local counters again to obtain a new value for the global counter. If this value is smaller than the previous one, then a wraparound has occurred, and p_i jumps directly to a place in the code that handles this situation. Note that a wraparound may occur, but the global counter's value obtained by the procedure is greater than the earlier value of the global counter and the wraparound failure is not detected. We will show

check-wraparound$_i$(*counter*)
1: *temp* := $\sum_i S_i.counter \bmod m$
2: **if** *temp* $\leq$ *counter* **then goto** Wraparound code

FIG. 6. *The check-wraparound*$_i$ *procedure.*

that when a wraparound failure occurs but is not detected, a takeover failure must be detected by the handshake mechanism.

4.2.4. Data structures. For simplicity, we assume that the shared memory consists of only n single-writer multireader registers, $R_1, \ldots, R_n$. All the information written by processor p_i is stored in its register R_i, which contains the following fields:

S_i. p_i's segment, with three fields: *value*, (unbounded) *seq*, and (modulo m) *counter*.

$Tree_i$. p_i's registers in the classification tree of the m-shot object (one register per node). Each register holds the same three fields as above.

$H_{i,1}, \ldots, H_{i,n}$. The handshake variables of p_i with respect to all of the other processors. For simplicity, we assume p_i holds handshake variables also with respect to itself.

$Last[1,2]$. $Last[1]$ holds the view returned by the last scate operation by p_i. $Last[2]$ holds the view returned by the penultimate scate operation by p_i.

In the code and throughout the correctness proof, we refer to the various fields of the registers $R_1, \ldots, R_n$ separately. Any *read* operation from some field of a register implies that the whole register is read. Any *write* operation to some field means writing some new value to that specific field and rewriting the current values to the other fields.

4.2.5. Code description. The code appears in Figure 7.

In the code, p_i starts by recording the sequence number of its last operation and then incrementing its local sequence number and counter variables. Then, p_i performs the handshake procedure for each processor and then calculates the global counter. At this point, p_i writes the value of the operation into its segment S_i. Notice that it is possible that this line is not executed at all, since p_i may detect a takeover failure while collecting the values of the local counters (in line 4). In this case, p_i jumps directly to line 17 to handle the takeover failure and writes the value of the operation into S_i there. (Failure handling is explained later.) p_i proceeds by performing a wraparound check. If a wraparound is detected, p_i jumps to line 23. If no wraparound is detected, p_i collects a local view of the segments and starts to traverse the classification tree. This part of the operation is performed almost exactly as in the m-shot snapshot object, except that the calculations regarding the sizes of views, performed in the *classifier* procedures, are done modulo m. If p_i traverses the tree without detecting any takeover failure, it obtains some temporary result. Then, p_i performs another wraparound detection procedure. If during this procedure no failure is detected, p_i returns the temporary result as the result of the operation (and updates $R_i.Last[1,2]$). Otherwise, p_i jumps to handle the detected failure.

Both kinds of failures, takeover and wraparound, are handled in a similar manner. When p_i detects that it was overtaken by p_j, it tries to copy p_j's last returned view. However, p_i is allowed to do so only if the last view returned by p_j contains p_i's current operation value. If not, p_i starts the operation all over again. When p_i detects a wraparound failure, it tries to find a sufficiently recent view that was returned by some operation and copies it. More precisely, p_i tries to find a penultimate view of

```
scate(value)      (Code for p_i)
0:  first-seq := sequence-number

    Start:
1:  sequence-number := sequence-number+1
2:  my-counter := (my-counter + 1) mod m
3:  for j = 1 to n do H_{i,j} := Handshake_i(j)
4:  g-counter := Σ_i S_i.counter mod m
5:  S_i := (value,sequence-number,my-counter)
6:  check-wraparound(g-counter)
7:  in_i := read S_1, ..., S_n
8:  v := root, current_{i,1} := in_i
9:  for ℓ = 1... log m do
10:     current_{i,ℓ+1} := Classifier(Label(v), current_{i,ℓ})
11:     if master then v := v.right
12:     if slave then v := v.left
13: temp-result := current_{i,log m+1}
14: check-wraparound(g-counter)
15: R_i.Last[1,2] := ⟨temp-result, R_i.Last[1]⟩
16: return temp-result

    Takeover by p_j code:
17: S_i := (value,sequence-number,my-counter)
18: temp-result := R_j.Last[1]
19: if temp-result[i].seq > first-seq then
20:     R_i.last[1,2] := ⟨temp-result, R_i.Last[1]⟩
21:     return temp-result
22: else goto Start

    Wraparound code:
23: if ∃R_j.Last[2][i].seq > first-seq then
24:     R_i.Last[1,2] := ⟨R_j.Last[1], R_i.Last[1]⟩
25:     return R_j.Last[1]
26: else goto Start
```

FIG. 7. *The scate operation.*

some processor that contains p_i's current operation value. If p_i finds such a processor, it copies its last view; otherwise, p_i starts the operation all over again.

As a consequence of the failure handling technique, a scate operation may consist of several *attempts*. (Each time a processor arrives at the label Start is the beginning of a new attempt.) For every scate operation, only its last attempt is successful and returns a view. The successful attempts can either return a view through the failure handling procedures or not. Therefore, we partition attempts into three types: *unsuccessful* attempts, which do not return a view; *indirect* attempts, which return a copied view in line 21 or 25; and *direct* attempts, which return a view in line 16.

Note that different attempts of the same operation have different sequence numbers. Therefore, the unsuccessful attempts may be thought of as independent operations that are "cut off" before completion. On the other hand, the same *first-seq* is used by all attempts of the same operation. The value of *first-seq* is used in order to

decide whether to copy another processor's view in the failure procedures. That is, the conditions in lines 19 and 23 are satisfied if the found view contains the sequence number of any of the attempts of the current operation.

4.3. Correctness proof. We first show that views returned by scate operations are comparable. Since only successful attempts return views, it suffices to prove comparability for them.

Define an ordering on attempts according to the order they update the segments. (This order has nothing to do with the linearization of scans and updates which will be presented later.) Specifically, for each attempt we consider the first time that it writes to S_i, either in line 5 or in line 17. This write is called the *actual update* of the attempt. Since writes are atomic, the ordering of actual updates defines an ordering among the attempts.

Based on the ordering of the attempts, we divide them into "virtual rounds" of size m. The first round contains the first m attempts, and in general, the ith round contains attempts $(i-1)m+1, \ldots, im$.

Recall that k is the constant for the handshake mechanism, and m is a constant that determines the height of the classification tree. These constants were left unspecified, and we now fix $k = 8$ and $m = (k+2)n = 10n$.

The following lemma implies that in order to prove the comparability of views returned by successful attempts, we can consider only the direct attempts.

LEMMA 4.1. *A view returned by an indirect attempt is also returned by some direct attempt.*

Proof. Toward a contradiction, let at_i be an indirect attempt that copies a view from some $R_j.Last[1]$ such that this view is not a direct view. Consider all the attempts that return the same view as at_i, and from these attempts let at_k be the attempt whose write before returning its view (in lines 20 or 24) is the first in the execution. The view returned by at_k must be direct; otherwise, there was some other attempt that returned this view and wrote it before at_k did, which is a contradiction. □

We next show that the views returned by direct attempts can be organized by the virtual rounds.

LEMMA 4.2. *Let at_i be a direct attempt in round r_i, and let at_j be a (direct or indirect) attempt in round $r_j > r_i$. Then at_i starts to execute its wraparound test in line* 14 *before at_j executes its actual update step.*

Proof. We slightly abuse notation and denote the processors that execute at_i and at_j by p_i and p_j, respectively. Note that p_i and p_j may be the same processor, while at_i and at_j are not the same attempt. This should not cause any confusion.

Consider the execution of line 4 in at_i, and let c be the value of *g-counter*. Since at_i is in round r_i, the value of the global counter is still less than $(r_i+1)m$ when p_i completes line 4. Now p_i executes its actual update step. Since at_i is direct, p_i continues without detecting any failure and arrives in line 14.

Assume, by way of contradiction, that p_j executes its actual update step in at_j before p_i starts line 14. Therefore, the value of the global counter is greater than $(r_i+1)m$ when p_i starts line 14, since at_j is in round $r_j > r_i$. Since p_i does not detect a wraparound in line 14, the value it reads is $c' \geq c$. This can happen only if the local counters were incremented at least $m = (k+2)n$ times since p_i started to execute line 4. In particular, at least one processor p_l incremented its counter at least $(k+2)$ times since p_i has started to execute line 4. Thus, p_l performs $handshake_l(i)$ at least $(k+1)$ times since p_i started to execute line 4, which implies that p_l performs

handshake$_l(i)$ at least $(k+1)$ times since p_i performed *handshake*$_i(l)$ in at_i. By the properties of the handshake mechanism, p_i will detect a takeover failure by p_l while executing line 14, which is a contradiction. □

This implies the following corollary.

COROLLARY 4.3. *Let at_i be a direct attempt in round r_i. The view returned by at_i does not contain any values written by attempts in rounds strictly greater than r_i.*

By the definition of rounds, when p_i reads $S_1, \ldots, S_n$ in line 7 it observes all the values from previous rounds. Furthermore, it is immediate from the code that any direct attempt returns a view which contains at least the values it reads in line 7. Therefore, we have the following corollary.

COROLLARY 4.4. *Let at_i be a direct attempt of round r_i. The view returned by at_i contains all the values written in rounds strictly smaller than r_i.*

The above corollaries indicate that a direct attempt in round r observes all the values of rounds smaller than r, plus some subset of the values of round r, and nothing from rounds greater than r. Thus, for any two direct attempts in different rounds, it is clear that the view returned by the later attempt dominates the view returned by the earlier one. Consequently, in order to prove comparability of all the direct views, we need only prove comparability of attempts in the same round. This is done in the next lemma.

LEMMA 4.5. *Let at_i and at_j be two direct attempts of round r. The views returned by at_i and at_j are comparable.*

Proof. By Lemma 4.2, until both at_i and at_j arrive at line 14, no value of round greater than r is written in the segments and certainly not in the registers of the tree. In addition, when either at_i or at_j reads the segments before starting to traverse the tree (at line 7), all $(r-1)m$ values of rounds $1, \ldots, r-1$ are already written in the segments. Thus, the contribution of these values to the calculations that are performed in the *classifier* procedures that are executed throughout at_i and at_j is cancelled out.

This implies that the process of traversing the tree by at_i and at_j has exactly the same properties of the m-shot object construction. The comparability of the views returned by at_i and at_j is implied by the same arguments as in the m-shot object (in the proofs of Lemmas 3.4 and 3.6). □

Combining the above lemma with Lemma 4.1 implies the following corollary.

COROLLARY 4.6. *The views returned by any two scate operations are comparable.*

Comparable scate operations are used to implement scans and updates exactly in the same way as in the m-shot object. That is, to execute an update(v) operation, a processor executes scate(v) operation and ignores the value it returns; to execute a scan operation, a processor executes a scate(v) operation with the current value of its segment.

We now linearize the scan and update operations. First we identify each (update or scan) operation with the unique pair (v, seq) that is written by the first attempt of the operation. Scans and updates are linearized as in the m-shot object. That is, the scans are linearized according to the (comparable) views they return, and the updates are linearized between the scans according to the values they write. Clearly, by the way updates are linearized between scans, we have the following lemma.

LEMMA 4.7. *For every scan sc and for every S_i, the value returned by sc for S_i is the value written by the last update by p_i that is linearized before sc.*

Therefore, the sequence preserves the semantics of the snapshot object. To show it is a linearization, it remains to prove that the above sequence is consistent with the real time order of operations, $\rightarrow$.

The proof is similar to the corresponding proof for the m-shot object, but it is more complicated since in the m-shot object all the returned views were direct, while here the proof must consider both direct and indirect views. We start by introducing some terminology.

We say that an operation *op* (scan or update) returns a direct view if the successful attempt of *op* is direct, and similarly for indirect view. In addition, we sometimes classify *op* itself as direct or indirect.

The *origin* of an operation *op* is the attempt that directly returned the view returned by *op*. Formally, the origin of an operation *op* is defined inductively as follows. If *op* is direct, then the origin of *op* is the last attempt executed in *op*. Otherwise, if *op* is indirect and copies the view returned by op', then the origin of *op* is the origin of op'. In a similar manner, we define the *depth* of an operation *op*, which specifies the distance of *op* from its origin. If *op* is direct, then its depth is zero. Otherwise, if *op* is indirect and copies the view returned by op', then the depth of *op* equals the depth of op' plus one.

An *interval* is a subsequence of consecutive primitive operations in the execution. The *interval of an operation* is the interval starting with the execution of the first statement of the operation and ending with the execution of the last statement of the operation (not including the **Return** statement). The *interval of an attempt* is defined similarly.

An interval is *unsafe* if some processor starts and terminates two consecutive unsuccessful attempts in this interval. Otherwise, the interval is *safe*.

To show that the sequence defined above is consistent with $\rightarrow$, it suffices to prove that any indirect operation starts before its origin. This implies that the view copied from the origin is sufficiently up to date, and thus, the indirect operation is linearized within its interval. The intuitive proof argues that if an operation fails (due to either takeover or wraparound), then during the time interval of the operation many other operations were performed. At least some of these operations are completely contained in the interval, and therefore, the view copied by the indirect operation must be sufficiently up to date.

Unfortunately, the above intuition is not accurate since the failure mechanisms guarantee only that during the interval of an indirect operation there are many *attempts*. However, it is possible that not many of the attempts are successful, and therefore, not many operations are completed during this interval. This means that there are no up to date views to be copied. To overcome this problem we must show that an operation does not contain many attempts. This will imply that if there are many attempts in some interval, then there are many operations as well. To prove that an operation does not contain many attempts, we have to show that after a small number of unsuccessful attempts, an operation will find its value in some already existing view (or penultimate view). In turn, this relies on the fact that when a failure is detected, there are sufficiently up to date views that were obtained by other operations. On the face of it, this argument seems circular.

Put another way, the difficulty arises because the proof of partial correctness (processors return values that are up to date) relies on the assumption that operations terminate, and vice versa. We sidestep this circularity by first proving partial correctness if the operation's interval is safe, that is, all operations inside it terminate after (at most) two attempts. Using this fact, we then prove that any interval is safe, i.e., all operations terminate after (at most) two attempts. This implies that the claim holds for any operation.

LEMMA 4.8. *If op's interval is safe, then op's origin starts after op starts.*

Proof. The proof is by induction on d, the depth of op. The base case, $d = 0$, follows since the last attempt of op is the origin of op. For the induction step, let op be an operation with depth $d > 0$, and assume the lemma holds for any operation of depth $d - 1$ whose interval is safe. Since $d > 0$, op is indirect, and it copies the view of some operation op' of processor p' with depth $d - 1$. Let at and at' denote the successful attempts of op and op', respectively. There are two cases.

Case 1. op copies the view of op' due to a takeover failure. Since takeover failures are detected by the handshake procedure, p' has executed its handshake procedure at least $k \geq 6$ times while at was executed. Therefore, p' starts and completes at least four consecutive attempts during at's interval. Since at's interval is safe, at least two of these attempts are successful. Therefore, p' completes at least two operations while at is executed. The attempt at copies the view returned by op', which is the last preceding view returned by p'. The above implies that op' starts after at starts. By the induction hypothesis, the origin of op' starts after op' starts. Since this is also the origin of op, it follows that the origin of op starts after op starts.

Case 2. op copies the view of op' due to a wraparound failure. Let op'' be the operation of p' that precedes op'. By the condition for copying the view of op', the view returned by op'' contains the value written by op. Therefore, op'' does not terminate before op starts. In particular, op' starts after op starts. By the induction hypothesis, the origin of op' starts after op' starts. Since this is also the origin of op, it follows that the origin of op starts after op starts. □

We now prove that all intervals are safe, by showing that every operation terminates after at most two attempts.

LEMMA 4.9. *Every operation contains at most two attempts.*

Proof. Assume, by way of contradiction, that there is an operation op_i by processor p_i that contains two consecutive unsuccessful attempts, at_1, at_2. Assume that the interval from the start of at_1 to the completion of at_2 is minimal, that is, all intervals contained in it are safe. (Such a minimal interval exists because the execution is a sequence.) We prove that at_2 must be successful. There are two cases.

Case 1. at_2 fails due to a takeover failure by processor p_j. In this case, p_j executes its handshake procedure at least $k \geq 6$ times during at_2's interval. This implies that in this interval p_j starts and completes at least four attempts. Since any interval strictly contained in at_2's interval is safe, at least two of these attempts are successful. Let op_j be the last operation completed by p_j in at_2's interval. It follows that op_j starts after at_2 starts, and therefore after the actual update of op_i to S_i (since at_2 is not the first attempt of op_i). Since op_j's interval is safe, Lemma 4.8 implies that op_j's origin starts after op_j starts, and therefore after the value of op_i is written in S_i. This implies that the view returned by op_j contains the value written by op_i. Therefore, when p_i discovers a takeover by p_j in at_2, it can copy the view of op_j, and hence at_2 is successful, which is a contradiction.

Case 2. at_2 fails due to a wraparound failure. Consider the interval from the start of at_1 to the completion of at_2. If at_1 is unsuccessful due to a takeover failure, then clearly there is a processor p_j that executes its handshake procedure at least $k \geq 8$ times during this interval. Otherwise, if both at_1 and at_2 fail due to a wraparound failure, then again it is guaranteed that during their interval there is a processor p_j that executes its handshake procedure at least $k \geq 8$ times. This implies that in this interval p_j starts and completes at least six attempts. Since this interval is safe, at least three of these attempts are successful. This implies that p_j starts and completes at least two operations in this interval. As before, since this interval is safe, Lemma 4.8 implies that the last two operations of p_j in this interval return views that contain

the value written by op_i. Therefore, when p_i discovers a wraparound failure in at_2, it can copy the last view returned by p_j, and hence at_2 is successful, which is a contradiction. □

Thus, all operation intervals are safe, and therefore Lemma 4.8 can be applied to any operation to obtain the following corollary.

COROLLARY 4.10. *For any operation op, the origin of op starts after op starts.*

This implies that indirect operations copy views which are up to date. Since direct operations clearly observe the value they write, and since indirect operations copy other processors' view only if it includes their value, we have the following lemma.

LEMMA 4.11. *Any scan or update operation returns a view that contains its own value.*

The following lemma proves that the linearization sequence preserves the real time order of the operations.

LEMMA 4.12. *For any two (scan/update) operations op_i and op_j, if $op_i \rightarrow op_j$ then op_i is linearized before op_j.*

Proof. There are four cases, according to operation types.

Case 1. Let sc_i and sc_j be two scan operations such that $sc_i \rightarrow sc_j$. By Lemma 4.11, sc_j returns a view that contains the value it writes. Furthermore, sc_i does not return a view that contains the value of sc_j. Since the views returned by sc_i and sc_j are comparable (Corollary 4.6), it must be that the view returned by sc_j dominates the view returned by sc_i. Therefore, sc_i is linearized before sc_j.

Case 2. Let sc_i be a scan operation and up_j be an update operation such that $sc_i \rightarrow up_j$. By the code of the algorithm, the view returned by sc_i does not contain the value written by up_j, and therefore up_j is linearized after sc_i.

Case 3. Let up_i be an update operation and sc_j be a scan operation such that $up_i \rightarrow sc_j$. By Corollary 4.10, the origin of sc_j starts after sc_j does, and therefore after up_i's actual update. Since the origin is a direct attempt, it reads up_i's value. Therefore, sc_j returns a view that contains the value written by up_i, and hence sc_j is linearized after up_i.

Case 4. Let up_i and up_j be two update operations such that $up_i \rightarrow up_j$. If up_i and up_j fit exactly between the same two scan operations, then due to the way symmetry is broken, up_i is linearized before up_j, and the lemma follows.

Otherwise, if up_j is linearized before up_i, then there exists a scan operation sc such that up_j is linearized before sc and sc is linearized before up_i. Thus, sc returns a view that contains the value written by up_j and does not contain the value written by up_i. Consider the scate operation that is executed to implement up_i. This scate operation returns a view that contains the value written by up_i (Lemma 4.11) but does not contain the value written by up_j (since $up_i \rightarrow up_j$). Therefore, this scate operation returns a view that is incomparable to the view returned by sc. This contradicts the comparability property of the views returned by scate operations (Corollary 4.6). □

Lemmas 4.7 and 4.12 imply that the sequence of scans and updates defined above is indeed a linearization. By Lemma 4.9, each scate operation contains at most two attempts. Each attempt requires $O(n \log m) = O(n \log n)$ operations on atomic single-writer multireader registers, which implies the following lemma.

LEMMA 4.13. *Any scan or update operation terminates after at most $O(n \log n)$ operations on atomic single-writer multireader registers.*

Note that, in addition to a single copy of the m-shot classification tree, each processor maintains n handshake variables (each with $O(k)$ possible values) and two views. Since k is a constant and $m = O(n)$, the algorithm requires a total of $O(n^2 B)$

bits, where as before, B is the number of bits required for representing a view. Note that B is still unbounded, since the algorithm still uses unbounded sequence numbers.

4.4. Bounding the sequence numbers. So far, we presented the ∞-shot snapshot object using unbounded sequence numbers to allow every processor to distinguish, for any set of values of another processor, which one is the most up to date. When sequence numbers are unbounded this goal is easily achieved by choosing the value with the maximal sequence number. To avoid unbounded values we use *bounded sequential timestamps*, a concept introduced in [20]. In our case, each processor generates its own set of timestamps (timestamps of different processors are not compared). Therefore, we can use ideas of [14] to implement these timestamps. Below, we briefly describe these ideas; the reader is referred to [14, 13] for further details.

The main idea is to allow a processor to know which of its sequence numbers might be in use in the system. If this can be done, then a processor can simply choose a new sequence number to be some value that is not in use; to let other processors know what is the ordering among its sequence numbers, the processor holds an ordered list of all its currently used sequence numbers. If the number of sequence numbers that might be in use is bounded, then the processor can draw its sequence numbers from a bounded set of values (thus effectively recycling them).

The difficult part of the above idea is keeping track of the sequence numbers that are in use in the system. The natural idea that comes to mind is that all of the sequence numbers of a processor that are written somewhere in the shared memory at some point are the ones that are in use. However, there might be situations where some processor, p_i, reads a certain sequence number, x, and then x is overwritten and "disappears" from the shared memory. Later on, p_i might rewrite x in the shared memory. The *traceable use* abstraction of [14] solves this problem of "hidden" values by forcing a processor that reads a sequence number from the shared memory to leave evidence that this sequence number was read. This results in a slightly more complicated mechanism for reading and writing values from the shared memory.

To allow values to be recycled the processor invokes a "garbage collection" of sequence numbers, whose execution is spread over the duration of several operations (see further details in [14, 13]).

The number of (low-level read and write) operations required for generating bounded sequence numbers is linear, and therefore the $O(n \log n)$ complexity of the snapshot algorithm is not affected.

In the implementation of the traceable use abstraction, the number of sequence numbers that each processor uses is bounded by $O(n^2)$ times the total number of sequence numbers of that processor that may be in the system concurrently (cf. [13]). In our case, each processor can have at most $O(n^2)$ of its own sequence numbers in the system concurrently. Thus, the total number of sequence numbers that are used by each processor is $O(n^4)$, and the size of the sequence numbers is therefore $O(\log n)$. In addition, each processor must hold an ordered list of all its sequence numbers that are currently in use. The list requires $O(n^4 \log n)$ bits per processor.

As was calculated before, the algorithm requires $O(n^2 B)$ bits, where B is the number of bits required to represent a view. To calculate B, recall that a view contains n entries, each with three fields: the actual value of the entry, the *counter* field ($O(\log n)$ bits), and the *seq* field (now bounded to require $O(\log n)$ bits). Therefore,

the number of bits required to represent a view is $O(n \log n)$, plus n times the number of bits required to represent an actual values of the snapshot object, which we denote by $|V|$. Thus, the total space complexity is $O(n^3(\log n + |V|) + n^5 \log n)$ bits.

5. Discussion. We introduced an implementation of a bounded atomic snapshot object in which each update or scan operation requires $O(n \log n)$ operations on atomic single-writer multireader registers. (As was previously mentioned, one of the operations can be made linear by the results of [23].) Obviously, at least $\Omega(n)$ operations are required for implementing the scan operation for an atomic snapshot object, and by [22] this is also the lower bound for implementing the update operation. Needless to say, it will be very interesting to close the $O(\log n)$ gap between our implementation and this lower bound.

REFERENCES

[1] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit, *Atomic snapshots of shared memory*, J. ACM, 40 (1993), pp. 873–890.

[2] J. H. Anderson, *Composite registers*, Distrib. Comput., 6 (1993), pp. 141–154.

[3] J. Aspnes, *Time- and space-efficient randomized consensus*, in Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing, ACM, New York, 1990, pp. 325–331.

[4] J. Aspnes and M. P. Herlihy, *Wait-free data structures in the asynchronous PRAM model*, in Proceedings of the 2nd Annual Symposium on Parallel Algorithms and Architectures, ACM, New York, 1990, pp. 340–349.

[5] H. Attiya, A. Bar-Noy, and D. Dolev, *Sharing memory robustly in message-passing systems*, in Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing, ACM, New York, 1990, pp. 363–376.

[6] H. Attiya, D. Dolev, and N. Shavit, *Bounded polynomial randomized consensus*, in Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing, 1989, ACM, New York, pp. 281–293.

[7] H. Attiya, M. Herlihy, and O. Rachman, *Atomic snapshots using lattice agreement*, Distrib. Comput., 8 (1995), pp. 121–132.

[8] H. Attiya, N. A. Lynch, and N. Shavit, *Are wait-free algorithms fast?*, J. ACM, 41 (1994), pp. 725–763.

[9] T. Chandra and C. Dwork, *Using Consensus to Solve Atomic Snapshots*, 1992, manuscript.

[10] D. Dolev and N. Shavit, *Bounded concurrent time-stamping*, SIAM J. Comput., 26 (1997), pp. 418–455.

[11] C. Dwork, private communication.

[12] C. Dwork, M. P. Herlihy, S. A. Plotkin, and O. Waarts, *Time-lapse snapshots*, in Proceedings of the Israel Symposium on the Theory of Computing and Systems, Haifa, Israel, 1992, Lecture Notes in Comput. Sci. 601, D. Dolev, Z. Galil, and M. Rodeh, eds., Springer-Verlag, Berlin, pp. 154–170.

[13] C. Dwork, M. P. Herlihy, and O. Waarts, *Bounded round numbers*, in Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing, ACM, New York, 1993, pp. 53–64.

[14] C. Dwork and O. Waarts, *Simple and efficient concurrent timestamping or bounded concurrent timestamping are comprehensible*, in Proceedings of the 24th ACM Symposium on Theory of Computing, ACM, New York, 1992, pp. 655–666.

[15] R. Gawlick, N. Lynch, and N. Shavit, *Concurrent timestamping made simple*, in Proceedings of the Israel Symposium on the Theory of Computing and Systems, Haifa, Israel, 1992, Lecture Notes in Comput. Sci. 601, D. Dolev, Z. Galil, and M. Rodeh, eds., Springer-Verlag, Berlin, pp. 171–183.

[16] M. P. Herlihy, *Wait-free synchronization*, ACM Trans. Programming Lang. Systems, 13 (1991), pp. 124–149.

[17] M. P. Herlihy, *Randomized wait-free objects*, in Proceedings of the 10th ACM Symposium on Principles of Distributed Computing, ACM, New York, 1991, pp. 11–21.

[18] M. P. Herlihy and J. M. Wing, *Linearizability: A correctness condition for concurrent objects*, ACM Trans. Programming Lang. Systems, 12 (1990), pp. 463–492.

[19] M. Inoue, W. Chen, T. Masuzawa, and N. Tokura, *Linear-time snapshot using multi-writer multi-reader registers*, in Proceedings of the 8th International Workshop on Distributed Algorithms, Terschelling, The Netherlands, 1994, Lecture Notes in Comput. Sci. 857, G. Tel and P. Vitanyi, eds., Springer-Verlag, Berlin, 1994, pp. 130–140.

[20] A. Israeli and M. Li, *Bounded time stamps*, Distrib. Comput., 6 (1987), pp. 205–209.

[21] A. Israeli and A. Shirazi, *Efficient Snapshot Protocol Using* 2*-Lattice Agreement*, 1992, manuscript.

[22] A. Israeli and A. Shirazi, *The time complexity of updating snapshot memories*, in 2nd Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 855, Springer-Verlag, New York, 1994, pp. 171–182.

[23] A. Israeli, A. Shaham, and A. Shirazi, *Linear-time snapshot protocols for unbalanced systems*, in Proceedings of the 7th International Workshop on Distributed Algorithms, A. Schiper, ed., Lausanne, Switzerland, 1993, Lecture Notes in Comput. Sci. 725, Springer-Verlag, Berlin, 1993, pp. 26–38.

[24] L. M. Kirousis, P. Spirakis, and Ph. Tsigas, *Reading many variables in one atomic operation: Solutions with linear or sublinear complexity*, in Proceedings of the 5th International Workshop on Distributed Algorithms, Delphi, Greece, 1991, Lecture Notes in Comput. Sci. 579, S. Toueg, P. Spirakis, and L. Kirousis, eds., Springer-Verlag, Berlin, 1991, pp. 229–241.

SIAM J. COMPUT.
Vol. 27, No. 2, pp. 341–355, April 1998
© 1998 Society for Industrial and Applied Mathematics
002

CIRCUIT BOTTOM FAN-IN AND COMPUTATIONAL POWER*

LIMING CAI†, JIANER CHEN‡, AND JOHAN HÅSTAD§

Abstract. We investigate the relationship between circuit bottom fan-in and circuit size when circuit depth is fixed. We show that in order to compute certain functions, a moderate reduction in circuit bottom fan-in will cause significant increase in circuit size. In particular, we prove that there are functions that are computable by circuits of linear size and depth k with bottom fan-in 2 but require exponential size for circuits of depth k with bottom fan-in 1. A general scheme is established to study the trade-off between circuit bottom fan-in and circuit size. Based on this scheme, we are able to prove, for example, that for any integer c, there are functions that are computable by circuits of linear size and depth k with bottom fan-in $O(\log n)$ but that require exponential size for circuits of depth k with bottom fan-in c, and that for any constant $\epsilon > 0$, there are functions that are computable by circuits of linear size and depth k with bottom fan-in $\log n$ but that require superpolynomial size for circuits of depth k with bottom fan-in $O(\log^{1-\epsilon} n)$. A consequence of these results is that the three input read-modes of alternating Turing machines proposed in the literature are all distinct.

Key words. computational complexity, circuit complexity, lower bound, alternating Turing machine

AMS subject classifications. 68Q05, 68Q10, 68Q15, 68Q25, 68Q30

PII. S0097539795282432

1. Introduction.

To prove lower bounds for various computational models remains as one of the most challenging tasks in complexity theory. Much progress has been made recently in deriving lower bounds for computational models with limited capabilities, with the hope that these may lead to better lower bounds for more general computational models and to better understanding of intrinsic complexity of computation.

One of the most successful trials is the derivation of lower bounds for constant depth circuits. The first strong lower bounds were given by Furst, Saxe, and Sipser [12] and, independently, by Ajtai [1] who show that the size of a constant depth circuit computing the parity function is superpolynomial. The results were subsequently sharpened by Yao [18] who derived an exponential lower bound. Håstad [14, 15] further strengthened the result and obtained near optimal lower bounds. A direct consequence of these results is that the logarithmic time hierarchy [17], i.e., the set of languages accepted by families of circuits of constant depth and polynomial size, is a proper subset of P.

The logarithmic time hierarchy was further refined by Sipser [17] who showed that for each integer $k > 1$, there are functions that are computable by a circuit of depth k and polynomial size but require superpolynomial size for circuits of depth $k - 1$. Thus, all levels of the logarithmic time hierarchy are distinct. Exponential

*Received by the editors March 6, 1995; accepted for publication (in revised form) December 19, 1995.

http://www.siam.org/journals/sicomp/27-2/28243.html

†School of Electrical Engineering and Computer Science, Ohio University, Athens, OH 45701 (cai@cs.ohiou.edu). The work of this author was supported in part by the Engineering Excellence Award from Texas A&M University.

‡Department of Computer Science, Texas A&M University, College Station, TX 77843-3112 (chen@cs.tamu.edu). The work of this author was supported in part by National Science Foundation grants CCR-9110824 and CCR-9613805.

§Department of Computer Science, Royal Institute of Technology, Stockholm, Sweden (johanh@nada.kth.se).

lower bounds for the depth k to $k-1$ conversion were first claimed by Yao [18] and then fully proved by Håstad [14, 15].

In this paper, we will further sharpen the separation results in the logarithmic time hierarchy by investigating the relationship between circuit bottom fan-in and circuit size when circuit depth is fixed. We show that in order to compute certain functions, a moderate reduction in circuit bottom fan-in will cause significant increase in circuit size. In particular, we prove that there are functions that are computable by circuits of linear size and depth k with bottom fan-in 2 but that require exponential size for circuits of depth k with bottom fan-in 1. A general scheme is established to study the trade-off between circuit bottom fan-in and circuit size. Based on this scheme, we are able to prove, for example, that for any integer c, there are functions that are computable by circuits of linear size and depth k with bottom fan-in $O(\log n)$ but that require exponential size for circuits of depth k with bottom fan-in c, and that for any constant $\epsilon > 0$, there are functions that are computable by circuits of linear size and depth k with bottom fan-in $\log n$ but that require superpolynomial size for circuits of depth k with bottom fan-in $O(\log^{1-\epsilon} n)$. Therefore, the computational power of constant depth circuits depends not only on its depth but also strictly on its bottom fan-in when the depth of the circuits is fixed.

Another motivation of our present research is the study of input read-modes of a sublinear-time alternating Turing machine, which is an important computational model in the study of complexity classes. A number of input read-modes for sublinear-time alternating Turing machines have appeared in the literature. In the standard model proposed by Chandra, Kozen, and Stockmeyer [9], a computation path of the machine can read up to $O(\log n)$ input bits in time $O(\log n)$. Ruzzo [16] proposed an input read-mode in which each computation path can read at most one input bit and the reading must be performed at the end of the path. An input read-mode studied by Sipser [17] insists that each input reading takes time $\Omega(\log n)$. These input read-modes have been carefully studied by Cai and Chen [6] who have given a precise circuit characterization for each read-mode for log-time alternating Turing machines of constant alternations. Input read-modes of log-time alternating Turing machines also find applications in the study of computational optimization problems [5, 8] and in the study of limited nondeterminism [7].

Based on the circuit characterizations of Cai and Chen and on our separation results in constant depth circuits, we are able to show that the three proposed input read-modes for alternating Turing machines are all distinct. More precisely, if we let $\Pi_k^{\mathcal{U}}$ (resp., $\Pi_k^{\mathcal{R}}$, $\Pi_k^{\mathcal{S}}$) be the class of languages accepted by log-time k-alternation alternating Turing machines using the input read-mode of Chandra, Kozen, and Stockmeyer (resp., Ruzzo, Sipser), then we can show that for all integers $k \geq 1$,

$$\Pi_k^{\mathcal{R}} \subset \Pi_k^{\mathcal{S}} \subset \Pi_k^{\mathcal{U}} \subset \Pi_{k+1}^{\mathcal{R}},$$

where $\subset$ means "proper subset." This gives a very detailed refinement of the logarithmic time hierarchy.

The paper is organized as follows. Section 2 introduces necessary definitions and related previous work. In section 3, we show that in order to compute certain functions, an $O(\log n)$ factor reduction in circuit bottom fan-in may cause exponential increase in circuit size. In section 4, we show that for circuits computing certain special functions, even reducing the circuit bottom fan-in by 1 will result in exponential

increase in circuit size. A general scheme is established in section 5 to study the trade-off between circuit bottom fan-in and circuit size. The relationship to the input read-modes of alternating Turing machines is given in section 6.

2. Preliminaries. We briefly review the fundamentals related to the present paper. For further discussion on the theory of circuit complexity and alternating Turing machines, the reader is referred to [3, 11].

An (unbounded fan-in) *Boolean circuit* α_n with input $x = x_1x_2, \ldots, x_n$ of length n is a directed acyclic graph. The *fan-in* of a node in the circuit is the in-degree of the node. The nodes of fan-in 0 are called *inputs* and are labeled from the set $\{0, 1, x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n\}$. The nodes of fan-in greater than 0 are called *gates* and are labeled either AND or OR. One of the nodes is designated the *output* node. The *size* is the number of gates and the *depth* is the maximum distance from an input to the output. Without loss of generality, we assume that the circuits are of the special form where all AND and OR gates are organized into alternating levels with edges only between adjacent levels. Any circuit may be converted to one of this form without increasing the depth and by at most squaring the size [10]. In this special form, the gates that are connected to input nodes will be called *bottom level gates* or *depth* 1 *gates*. The gates that receive inputs from depth 1 gates will be called *depth* 2 *gates*, and so on. The *bottom fan-in* of a circuit is the maximum over fan-ins of all bottom level gates. The following notation introduced by Boppana and Sipser [3] will be especially convenient in our discussion.

DEFINITION 2.1. (See [3].) *A circuit α is a Π_k^s-circuit (resp., Σ_k^s-circuit) if α is a depth k circuit of size at most s with an* AND*-gate (resp., an* OR*-gate) at the output. A circuit β is a $\Pi_k^{s,c}$-circuit (resp., $\Sigma_k^{s,c}$-circuit) if β is a depth $k+1$ circuit of size at most s with bottom fan-in c and an* AND*-gate (resp., an* OR*-gate) at the output.*

A *family* of circuits is a sequence $\{\alpha_n \mid n \geq 1\}$ of circuits, where α_n is with input of length n. A family of circuits may be used to define a language. A family $\{\alpha_n \mid n \geq 1\}$ of circuits is said to be a Π_k^{poly}-family (resp., $\Pi_k^{poly,c}$-family) if there is a polynomial p such that for all $n \geq 1$, α_n is a $\Pi_k^{p(n)}$-circuit (resp., $\Pi_k^{p(n),c}$-circuit).

The *Sipser function* f_k^m, as defined in [14, 15], is given by the circuit C_k^m shown in Figure 1. The circuit C_k^m is a tree of depth k in which every gate in the bottom level has fan-in $\sqrt{km \log m/2}$, the fan-in of the output gate is $\sqrt{m/\log m}$, and the fan-in for all other gates is m. The output gate of C_k^m is always an AND-gate, while the gates in the bottom level vary depending on the parity of k. Each variable x_i, $1 \leq i \leq n$, occurs at only one leaf. Note that the number n of variables of the function f_k^m equals $m^{k-1}\sqrt{k/2}$.

The following theorem is proved by Håstad [14, 15].

THEOREM 2.2. (See [14, 15].) *There is no depth k circuit computing the function f_k^m with bottom fan-in $\frac{1}{12\sqrt{2k}}\sqrt{\frac{m}{\log m}}$ and fewer than $2^{\frac{1}{12\sqrt{2k}}\sqrt{\frac{m}{\log m}}}$ gates of depth ≥ 2, for $m > m_0$, where m_0 is an absolute constant.*

To further sharpen the separation results and study the relationship between circuit bottom fan-in and circuit size, we introduce a variation $f_k^{m,b}$ of the Sipser function by explicitly specifying the bottom fan-in for the defining circuits.

DEFINITION 2.3. *Let $C_k^{m,b}$ be the tree circuit defining the Sipser function f_k^m, as illustrated in Figure 1, except that each bottom level gate of $C_k^{m,b}$ has fan-in b instead of $\sqrt{km \log m/2}$. Define $f_k^{m,b}$ to be the function computed by the tree circuit $C_k^{m,b}$. The number n of variables of the function $f_k^{m,b}$ is $n = bm^{k-2}\sqrt{m/\log m}$.*

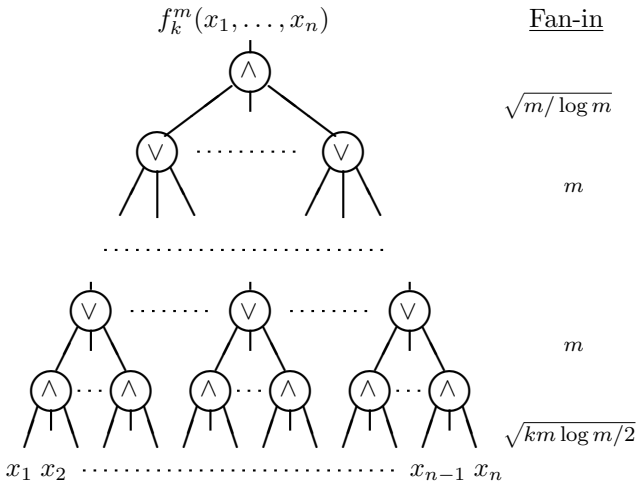


FIG. 1. *The circuit C_k^m defining the function f_k^m.*

The discussion of the present paper is centered on the complexity of the function $f_k^{m,b}$.

3. On circuits that compute $f_k^{m,\Omega(\log m)}$. In this section, we consider the complexity of the function $f_k^{m,b}$, where b is of order $\Omega(\log m)$.

Our main result in this section is that for $b > k \log m$, the function $f_k^{m,b}$ cannot be computed by any depth k circuit without exponential size and with bottom fan-in $\frac{b}{c \log m}$ for a particular constant c. This result requires two different proofs depending on whether the bottom fan-in b is larger or smaller than the bottom fan-in of the standard Sipser function circuit C_k^m given in Figure 1.

We first consider the case $b \leq \sqrt{km \log m/2}$. For this, we need to briefly review some notations and results by Håstad [14].

DEFINITION 3.1. (See [14].) *Let q be a real number and $(B_i)_{i=1}^r$ a partition of the variables. Let $R_{q,B}^+$ be the probability space of restrictions ρ that take values as follows.*

For every B_i, $1 \leq i \leq r$ independently,

1. *with probability q let $s_i = *$ and else $s_i = 0$.*
2. *for every $x_k \in B_i$ let $\rho(x_k) = s_i$ with probability q and else $\rho(x_k) = 1$.*

Similarly, an $R_{q,B}^-$ probability space of restrictions is defined by interchanging the roles played by 0 and 1.

The idea behind these restrictions is that a block B_i will correspond to the variables leading into one of the bottom level gates of the circuit $C_k^{m,b}$ that defines the function $f_k^{m,b}$. If the bottom level gates of $C_k^{m,b}$ are ANDs, we use a restriction from $R_{q,B}^+$; if the bottom level gates of $C_k^{m,b}$ are ORs, we use a restriction from $R_{q,B}^-$.

DEFINITION 3.2. (See [14].) *For a restriction $\rho \in R_{q,B}^+$, let $g(\rho)$ be the restriction defined as follows: for all B_i with $s_i = *$, $g(\rho)$ gives the value 1 to all variables which are given value $*$ by ρ except the variable with the highest index among those variables given value $*$ by ρ, to which $g(\rho)$ gives value $*$.*

If $\rho \in R_{q,B}^-$, then $g(\rho)$ is defined similarly but now takes the value 0 and $*$. Note that for a given restriction ρ, the restriction $g(\rho)$ can be obtained by a deterministic process that makes each block B_i have at most one $*$.

Let $\rho g(\rho)$ denote the composition of the two restrictions. That is, $\rho g(\rho)$ is the restriction $g(\rho)$ obtained from the restriction ρ.

LEMMA 3.3 (the switching lemma [14]). *Let σ be an* AND *of* ORs *all of size $\leq t$ and $\rho \in R^+_{q,B}$. Then the probability that under the restriction $\rho g(\rho)$ σ cannot be written as an* OR *of* ANDs *all of size $< s$ is bounded by α^s, where $\alpha < \frac{4qt}{\ln 2} < 5.78qt$.*

The switching lemma is also true if we do either or both of the following replacements: (1) replacing the probability space $R^+_{q,B}$ by the probability space $R^-_{q,B}$; and (2) replacing σ by an OR of ANDs to be converted to an AND and ORs.

Now we are ready for our first main result of this section.

THEOREM 3.4. *For $k \log m < b \leq \sqrt{km \log m/2}$, the function $f_k^{m,b}$ cannot be computed by any depth k circuit of bottom fan-in $\frac{b}{12k \log m}$ and size bounded by $2^{\frac{1}{12\sqrt{2(k-1)}}\sqrt{\frac{m}{\log m}}}$, for $m > m_0$, where m_0 is an absolute constant.*

Proof. The proof is similar to the induction step in the proof given by Håstad for Theorem 2.2 (see [14, pages 48–50]). Therefore, we only outline the proof and describe in detail those places that are different.

We set $q = \frac{k \log m}{b}$. Suppose that $\tau_1, \ldots, \tau_r$, $r = m^{k-2}\sqrt{m/\log m}$, are the bottom level gates of the tree circuit $C_k^{m,b}$ defining the function $f_k^{m,b}$. Let $(B_j)_{j=1}^r$ be the partition of the variables of $f_k^{m,b}$ such that block B_j is the set of variables leading into the bottom level gate τ_j of $C_k^{m,b}$.

Claim 1. The probability that under the restriction $\rho g(\rho)$ any bottom level gate τ_j of the tree circuit $C_k^{m,b}$ does not take value s_j is bounded by $\frac{1}{6m}$.

The proof is identical to the proof by Håstad for Fact 1 in [14, page 49]: such a gate τ_j does not take the corresponding value s_j with probability $(1-q)^{|B_j|} < \frac{1}{6}m^{-k}$. Since there are fewer than m^{k-1} bottom level gates in the tree circuit $C_k^{m,b}$, the probability in Claim 1 is bounded by $\frac{1}{6m}$.

Claim 2. The probability that under the restriction $\rho g(\rho)$ any depth 2 gate in the tree circuit $C_k^{m,b}$ gets fewer than $\sqrt{(k-1)m \log m/2}$ $*$'s from the bottom level is bounded by $\frac{1}{m}$.

Let $p_i = \binom{m}{i} q^i (1-q)^{m-i}$ be the probability that a fixed depth 2 gate μ in the tree circuit $C_k^{m,b}$ gets exactly i $*$'s from the bottom level. With the condition $b \leq \sqrt{km \log m/2}$, we can show that for $i \leq \sqrt{(k-1)m \log m}$, we have $\frac{p_i}{p_{i-1}} \geq \sqrt{2}$. Thus the probability that gate μ gets fewer than $\sqrt{(k-1)m \log m/2}$ $*$'s is bounded by m^{-k} for sufficiently large m. Since there are fewer than m^{k-1} depth 2 gates in the tree circuit $C_k^{m,b}$, the probability in Claim 2 is bounded by $\frac{1}{m}$.

Now suppose that the theorem is not true. Thus, there is a depth k circuit C_0 of size bounded by $2^{\frac{1}{12\sqrt{2(k-1)}}\sqrt{\frac{m}{\log m}}}$ and bottom fan-in $t \leq \frac{b}{12k \log m}$ that computes the function $f_k^{m,b}$. Furthermore, assume that the gates in the bottom level of the circuit C_0 are OR gates (the dual case can be proved similarly).

Claim 3. The probability that under the restriction $\rho g(\rho)$ any depth 2 gate in the circuit C_0 cannot be written as an OR of ANDs of size $\frac{1}{12\sqrt{2(k-1)}}\sqrt{\frac{m}{\log m}}$ is bounded by $\frac{1}{2}$.

Let σ be a fixed depth 2 gate in the circuit C_0. By the switching lemma, under the restriction $\rho g(\rho)$, the probability that σ cannot be written as an OR of ANDs of size $\frac{1}{12\sqrt{2(k-1)}}\sqrt{\frac{m}{\log m}}$ is bounded by $(5.78qt)^{\frac{1}{12\sqrt{2(k-1)}}\sqrt{\frac{m}{\log m}}}$. Since the circuit C_0 has at most $2^{\frac{1}{12\sqrt{2(k-1)}}\sqrt{\frac{m}{\log m}}}$ depth 2 gates, the probability in Claim 3 is bounded by

$$\begin{aligned}
&(5.78qt)^{\frac{1}{12\sqrt{2(k-1)}}\sqrt{\frac{m}{\log m}}} \cdot 2^{\frac{1}{12\sqrt{2(k-1)}}\sqrt{\frac{m}{\log m}}} \\
&< \left(\frac{5.78}{12}\right)^{\frac{1}{12\sqrt{2(k-1)}}\sqrt{\frac{m}{\log m}}} \cdot 2^{\frac{1}{12\sqrt{2(k-1)}}\sqrt{\frac{m}{\log m}}} \\
&\leq \left(\frac{5.78}{6}\right)^{\frac{1}{12\sqrt{2(k-1)}}\sqrt{\frac{m}{\log m}}},
\end{aligned}$$

which is smaller than $\frac{1}{2}$ when m is sufficiently large.

Therefore with a probability larger than $1 - (\frac{1}{6m} + \frac{1}{m} + \frac{1}{2}) > \frac{1}{3}$, the tree circuit $C_k^{m,b}$ becomes a circuit that computes a function f at least as hard as the Sipser function f_{k-1}^m, and the circuit C_0 becomes a depth $k-1$ circuit that computes the function f and has bottom fan-in $\frac{1}{12\sqrt{2(k-1)}}\sqrt{\frac{m}{\log m}}$ and fewer than $2^{\frac{1}{12\sqrt{2(k-1)}}\sqrt{\frac{m}{\log m}}}$ gates of depth ≥ 2. But this contradicts Theorem 2.2. □

Letting $b = \sqrt{km \log m/2}$ in Theorem 3.4, we obtain Theorem 2.2. Note that the size bound is slightly improved.

Note that the condition $b \leq \sqrt{km \log m/2}$ in Theorem 3.4 is essential in the proof for Claim 2. For larger bottom fan-in b, we have the following theorem.

THEOREM 3.5. *For $b \geq 2\sqrt{km \log m/2}$, the function $f_k^{m,b}$ cannot be computed by any depth k circuit of bottom fan-in $\frac{b}{25ke \log m}$ and size bounded by $2^{\frac{1}{12\sqrt{2k}}\sqrt{\frac{m}{\log m}}}$, for $m > m_0$, where m_0 is an absolute constant and e is the base of the natural logarithm.*

Proof. Let $q = \frac{1.04\sqrt{km \log m/2}}{b}$. Consider the following probability space R_q^+ of restrictions:

> For each variable x_k of the function $f_k^{m,b}$, let $\rho^+(x_k) = *$ with probability q and else $\rho^+(x_k) = 1$.

The probability space R_q^- is defined similarly except that the value 1 is replaced by value 0.

From now on, we assume that the bottom level gates of the tree circuit $C_k^{m,b}$ defining $f_k^{m,b}$ are AND gates. The case when the bottom level gates of $C_k^{m,b}$ are OR gates can be proved similarly by using the probability space R_q^- instead of the probability space R_q^+.

We first show that under a restriction $\rho^+ \in R_q^+$, with very large probability, the tree circuit $C_k^{m,b}$ computes a function at least as hard as the Sipser function f_k^m. The proof is similar to that for Claim 2 in Theorem 3.4. Thus, we only describe the differences.

Let τ be a bottom level gate in the tree circuit $C_k^{m,b}$. The gate τ is an AND gate of fan-in b. Let $p_i = \binom{b}{i} q^i (1-q)^{b-i}$ be the probability that the gate τ gets exactly i $*$'s under a restriction $\rho^+ \in R_q^+$. First we consider the ratio

$$\frac{p_i}{p_{i-1}} = \frac{b-i+1}{i} \cdot \frac{q}{1-q} > \frac{b-i}{i} \cdot \frac{q}{1-q}.$$

For $i \leq 1.02\sqrt{km \log m/2}$, we have

$$\frac{b-i}{i} \geq \frac{b - 1.02\sqrt{km \log m/2}}{1.02\sqrt{km \log m/2}}$$

and

$$\frac{q}{1-q} = \frac{(1.04\sqrt{km\log m/2})/b}{1-(1.04\sqrt{km\log m/2})/b} = \frac{1.04\sqrt{km\log m/2}}{b-1.04\sqrt{km\log m/2}}.$$

Thus, we have

$$\frac{p_i}{p_{i-1}} > \frac{b-1.02\sqrt{km\log m/2}}{1.02\sqrt{km\log m/2}} \cdot \frac{1.04\sqrt{km\log m/2}}{b-1.04\sqrt{km\log m/2}} \geq \frac{52}{51}.$$

This gives $(51/52)^{j-i}p_j > p_i$ for $i < j \leq 1.02\sqrt{km\log m/2}$.

Now under a restriction $\rho^+ \in R_q^+$, the probability that the gate τ gets fewer than $\sqrt{km\log m/2}$ $*$'s is bounded by

$$\begin{aligned}
&\sum_{i=0}^{\sqrt{km\log m/2}} p_i < \sum_{i=0}^{\sqrt{km\log m/2}} (51/52)^{\sqrt{km\log m/2}-i} p_{\sqrt{km\log m/2}} \\
\leq\ & 52 p_{\sqrt{km\log m/2}} < 52(51/52)^{0.02\sqrt{km\log m/2}} p_{1.02\sqrt{km\log m/2}} \\
\leq\ & 52(51/52)^{0.02\sqrt{km\log m/2}},
\end{aligned}$$

and $52(51/52)^{0.02\sqrt{km\log m/2}}$ is smaller than $\frac{1}{m^k}$ for sufficiently large m.

Since the circuit $C_k^{m,b}$ has fewer than m^{k-1} bottom level gates, we conclude that under a restriction $\rho^+ \in R_q^+$, the probability that any bottom level gate of the tree circuit $C_k^{m,b}$ gets fewer than $\sqrt{km\log m/2}$ $*$'s is bounded by $\frac{1}{m}$.

Now suppose that the theorem is not true. Thus, there is a depth k circuit C_0 of bottom fan-in at most $\frac{b}{25ke\log m}$ and size bounded by $2^{\frac{1}{12\sqrt{2k}}\sqrt{\frac{m}{\log m}}}$ such that the circuit C_0 computes the function $f_k^{m,b}$. We show that under a restriction $\rho^+ \in R_q^+$, with very large probability, the circuit C_0 becomes a depth k circuit of bottom fan-in at most $\frac{1}{12\sqrt{2k}}\sqrt{\frac{m}{\log m}}$.

Let μ be a bottom level gate of fan-in $c \leq \frac{b}{25ke\log m}$ in the circuit C_0, and let $r_i = \binom{c}{i}q^i(1-q)^{c-i}$ be the probability that the gate μ gets exactly i $*$'s under a restriction $\rho^+ \in R_q^+$. We have

$$\binom{c}{i}q^i(1-q)^{c-i} \leq \frac{c!}{i!(c-i)!}q^i \leq \frac{c^i q^i}{i!} \leq \left(\frac{cqe}{i}\right)^i,$$

where the last inequality is based on Stirling's approximation [13, page 452]

$$i! \geq 0.9(i/e)^i\sqrt{2\pi i} \geq (i/e)^i, \quad \text{for } i \geq 1.$$

Let $s = \frac{1}{12\sqrt{2k}}\sqrt{\frac{m}{\log m}}$. Under a restriction $\rho^+ \in R_q^+$, the probability that the gate μ gets more than s $*$'s is bounded by

$$\sum_{i=s+1}^{c} r_i \leq \sum_{i=s+1}^{c} \left(\frac{cqe}{i}\right)^i \leq \sum_{i=s+1}^{c} \left(\frac{\frac{1.04}{25\sqrt{2k}}\sqrt{\frac{m}{\log m}}}{i}\right)^i.$$

For $i > s = \frac{1}{12\sqrt{2k}}\sqrt{\frac{m}{\log m}}$, we have $\left(\frac{1.04}{25\sqrt{2k}}\sqrt{\frac{m}{\log m}}\right)\Big/ i < \frac{12.48}{25}$. Thus, under a restriction $\rho^+ \in R_q^+$ the probability that the gate μ gets more than s $*$'s is bounded by

$$\sum_{i=s+1}^{c} (12.48/25)^i < (12.48/25)^s.$$

Since the circuit C_0 has at most 2^s bottom level gates, we conclude that under a restriction $\rho^+ \in R_q^+$, the probability that any bottom level gate of the circuit C_0 gets more than s $*$'s is bounded by

$$(12.48/25)^s \cdot 2^s = (24.96/25)^s = (24.96/25)^{\frac{1}{12\sqrt{2k}}\sqrt{\frac{m}{\log m}}},$$

which is smaller than $\frac{1}{m}$ for sufficiently large m.

Thus, under a restriction $\rho^+ \in R_q^+$, with probability $\geq 1 - \frac{1}{m} - \frac{1}{m} > \frac{1}{2}$, all bottom level gates of the tree circuit $C_k^{m,b}$ get at least $\sqrt{km\log m/2}$ $*$'s (thus $C_k^{m,b}$ is converted to a circuit computing a function at least as hard as the Sipser function f_k^m), and all bottom level gates of the circuit C_0 get at most $\frac{1}{12\sqrt{2k}}\sqrt{\frac{m}{\log m}}$ $*$'s. Note that if a bottom level gate μ of the circuit C_0 gets at most $\frac{1}{12\sqrt{2k}}\sqrt{\frac{m}{\log m}}$ $*$'s, then either the gate μ is eliminated from the bottom level (e.g., μ is an AND gate and gets an input with value 0) or the gate μ becomes a gate of fan-in at most $\frac{1}{12\sqrt{2k}}\sqrt{\frac{m}{\log m}}$. In any case, we have derived that there is an assignment that converts the circuit C_0 into a depth k circuit C' of bottom fan-in bounded by $\frac{1}{12\sqrt{2k}}\sqrt{\frac{m}{\log m}}$ and size bounded by $2^{\frac{1}{12\sqrt{2k}}\sqrt{\frac{m}{\log m}}}$ such that the circuit C' computes a function at least as hard as the Sipser function f_k^m. But this contradicts Theorem 2.2.

This completes the proof. □

4. On circuits that compute $f_k^{m,2}$. In the previous section, we showed that for circuits to compute the function $f_k^{m,b}$, $b = \Omega(\log m)$, an $O(\log m)$ factor reduction in bottom fan-in may cause an exponential increase in the circuit size. In this section, we will show that in certain cases, even reducing the circuit bottom fan-in by 1 will cause an exponential increase in the circuit size. More precisely, we will show that the function $f_k^{m,2}$ can be computed by a depth k circuit of linear size and bottom fan-in 2 but requires exponential size for depth k circuits of bottom fan-in 1. Note that a depth k circuit of bottom fan-in 1 is actually a depth $k-1$ circuit.

We prove the above result with a new probability space of restrictions. We start with the following lemma.

LEMMA 4.1. *Partition the Boolean variables $\{x_1, \ldots, x_n\}$ into groups of c variables each. For each group, randomly pick r variables and assign them* 0*, and assign the rest $c-r$ variables $*$. Let σ be an* OR *of a subset S_σ of $\{x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n\}$ such that S_σ contains at least h negative literals $\overline{x}_i$. Then with the above random assignment,*

$$Pr[\sigma \not\equiv 1] \leq ((c-r)/c)^h.$$

Proof. Let $s = n/c$ be the number of groups in the partition given in the statement of the lemma. We first rename the literals in the set S_σ so that $\overline{x}_1^{(d)}, \ldots, \overline{x}_{j_d}^{(d)}$, $d = 1, \ldots, s$, $j_1 + \cdots + j_s = h$, are h negative literals in S_σ, where $\overline{x}_1^{(d)}, \ldots, \overline{x}_{j_d}^{(d)}$ belong to the same group in the partition, $d = 1, \ldots, s$.

Let $A_t^{(d)}$ be the event that the variable $x_t^{(d)}$ is *not* assigned 0 by the random assignment, and let $\mathcal{E}_{\sigma\not\equiv 1}$ be the event that σ is not identical to 1. Then

$$\mathcal{E}_{\sigma\not\equiv 1} \subseteq \bigcap_{d=1}^{s} \left(A_1^{(d)} \cap \cdots \cap A_{j_d}^{(d)} \right).$$

Thus,

$$Pr[\sigma \not\equiv 1] = Pr[\mathcal{E}_{\sigma\not\equiv 1}] \leq Pr\left[\bigcap_{d=1}^{s} \left(A_1^{(d)} \cap \cdots \cap A_{j_d}^{(d)} \right) \right] = \prod_{d=1}^{s} Pr\left[A_1^{(d)} \cap \cdots \cap A_{j_d}^{(d)} \right].$$

We have the last equality because the events $A_1^{(d)} \cap \cdots \cap A_{j_d}^{(d)}$ and $A_1^{(d')} \cap \cdots \cap A_{j_{d'}}^{(d')}$ are independent for $d \neq d'$.

Now consider $Pr[A_1^{(d)} \cap \cdots \cap A_{j_d}^{(d)}]$. If $j_d > c - r$, then by the way we assign the dth group, at least one of the variables $x_1^{(d)}, \ldots, x_{j_d}^{(d)}$ is assigned 0. Therefore,

$$Pr\left[A_1^{(d)} \cap A_2^{(d)} \cap \cdots \cap A_{j_d}^{(d)} \right] = 0.$$

If $j_d \leq c - r$, then

$$Pr\left[A_1^{(d)} \cap A_2^{(d)} \cap \cdots \cap A_{j_d}^{(d)} \right]$$
$$= Pr\left[A_1^{(d)} \right] \cdot Pr\left[A_2^{(d)} | A_1^{(d)} \right] \cdot Pr\left[A_3^{(d)} | A_1^{(d)} \cap A_2^{(d)} \right] \cdots Pr\left[A_{j_d}^{(d)} | A_1^{(d)} \cap \cdots \cap A_{j_d-1}^{(d)} \right].$$

Note that

$$Pr\left[A_i^{(d)} | A_1^{(d)} \cap \cdots \cap A_{i-1}^{(d)} \right] = \frac{c - r - i + 1}{c - i + 1} \leq \frac{c - r}{c}.$$

Thus,

$$Pr\left[A_1^{(d)} \cap A_2^{(d)} \cap \cdots \cap A_{j_d}^{(d)} \right] \leq ((c - r)/c)^{j_d}.$$

This gives directly

$$Pr[\sigma \not\equiv 1] \leq \prod_{d=1}^{s} Pr\left[A_1^{(d)} \cap \cdots \cap A_{j_d}^{(d)} \right] \leq \prod_{d=1}^{s} \left(\frac{c - r}{c} \right)^{j_d} = \left(\frac{c - r}{c} \right)^{h}. \qquad \square$$

Similarly we can prove the following lemma.

LEMMA 4.2. *Partition the Boolean variables $\{x_1, \ldots, x_n\}$ into groups of c variables each. For each group, randomly pick r variables and assign them 1, and assign the rest $c-r$ variables $*$. Let σ be an* OR *of a subset S_σ of $\{x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n\}$ such that S_σ contains at least h positive literals x_i. Then with the above random assignment,*

$$Pr[\sigma \not\equiv 1] \leq ((c - r)/c)^{h}.$$

Now we are ready for the main theorem of this section.

THEOREM 4.3. *The function $f_k^{m,2}$ cannot be computed by any depth $k-1$ circuit of size bounded by $2^{\frac{1}{12\sqrt{2(k-1)}}\sqrt{\frac{m}{\log m}}}$ for $m > m_0$, where m_0 is an absolute constant.*

Proof. To simplify the expressions, we let $s = \frac{1}{12\sqrt{2(k-1)}}\sqrt{\frac{m}{\log m}}$. Suppose that the theorem is not true and that there is a depth $k-1$ circuit C of size 2^s that computes the function $f_k^{m,2}$. Furthermore, we assume that the gates in the bottom level of C are OR gates (the case that the bottom level gates of C are AND gates can be proved similarly).

Randomly pick one variable from each pair x_{2i-1} and x_{2i} and assign it 0. This will reduce the tree circuit $C_k^{m,2}$ defining $f_k^{m,2}$ to the tree circuit $C_{k-1}^{m,m}$ defining $f_{k-1}^{m,m}$.

Let τ be an OR gate in the bottom level of the circuit C such that τ has more than s negative literals in its input, then by Lemma 4.1,

$$Pr[\tau \not\equiv 1] \leq (1/2)^{s+1}.$$

Let $\tau_1, \ldots, \tau_r$, $r \leq 2^s$ be all the gates in the bottom level of the circuit C such that there are more than s negative literals in their input, then

$$Pr[\tau_1 \not\equiv 1 \vee \cdots \vee \tau_r \not\equiv 1] \leq Pr[\tau_1 \not\equiv 1] + \cdots + Pr[\tau_r \not\equiv 1] \leq 2^s(1/2)^{s+1} = 1/2.$$

Thus,

$$Pr[\tau_1 \equiv 1 \wedge \cdots \wedge \tau_r \equiv 1] \geq 1/2.$$

Therefore, there is an assignment that converts the circuit $C_k^{m,2}$ to the circuit $C_{k-1}^{m,m}$ and eliminates all gates in the bottom level of the circuit C in whose input there are more than s negative literals. Let the circuit obtained from C by this assignment be C'.

Now partition the input of the function $f_{k-1}^{m,m}$ into groups of m variables each such that each group corresponds to the inputs to a bottom level gate of the tree circuit $C_{k-1}^{m,m}$. Randomly pick half of the variables in each group and assign them 1. The circuit $C_{k-1}^{m,m}$ under such an assignment is converted to the circuit $C_{k-1}^{m,m/2}$ defining the function $f_{k-1}^{m,m/2}$.

Let σ be an OR gate in the bottom level of the circuit C' with more than s positive literals in its input, then by Lemma 4.2,

$$Pr[\sigma \not\equiv 1] \leq (1/2)^{s+1}.$$

Let $\sigma_1, \ldots, \sigma_t$, $t \leq 2^s$ be all the gates in the bottom level of the circuit C' with more than s positive literals in their input, then

$$Pr[\sigma_1 \not\equiv 1 \vee \cdots \vee \sigma_t \not\equiv 1] \leq Pr[\sigma_1 \not\equiv 1] + \cdots + Pr[\sigma_t \not\equiv 1] \leq 2^s(1/2)^{s+1} = 1/2.$$

Thus,

$$Pr[\sigma_1 \equiv 1 \wedge \cdots \wedge \sigma_t \equiv 1] \geq 1/2.$$

Therefore, there is an assignment that converts the circuit $C_{k-1}^{m,m}$ to the circuit $C_{k-1}^{m,m/2}$ and eliminates all gates in the bottom level of C' that have more than s positive literals in their input. Let the circuit obtained from C' by this assignment be C''.

Since each gate in the bottom level of the circuit C'' has neither more than s negative literals nor more than s positive literals in its input, the bottom fan-in of the circuit C'' is at most $2s = \frac{1}{6\sqrt{2(k-1)}}\sqrt{\frac{m}{\log m}}$, which is smaller than $\frac{m/2}{25e(k-1)\log m}$ for

sufficiently large m. Thus, we have constructed a circuit C'' of depth $k-1$, bottom fan-in less than $\frac{m/2}{25e(k-1)\log m}$, and size bounded by $2^s = 2^{\frac{1}{12\sqrt{2(k-1)}}\sqrt{\frac{m}{\log m}}}$ such that C'' computes the function $f_{k-1}^{m,m/2}$. This contradicts Theorem 3.5. □

The following corollary will be used in section 6.

COROLLARY 4.4. *The function $f_k^{m,2}$ can be computed by a circuit of depth k, linear size, and bottom fan-in 2, but cannot be computed by any depth $k-1$ circuit of polynomial size.*

COROLLARY 4.5. *For each pair of integers $k, c > 1$, there are functions that are computable by circuits of linear size and depth k with bottom fan-in c but require exponential size for circuits of depth $k-1$.*

5. Trade-off between bottom fan-in and size. We first summarize the results in the previous two sections in the following theorem.

THEOREM 5.1. *For all integers $b \geq 2$ and sufficiently large m, the function $f_k^{m,b}$ can be computed by a depth k circuit of linear size and bottom fan-in b, but requires size larger than $2^{\frac{1}{12\sqrt{2k}}\sqrt{\frac{m}{\log m}}}$ for depth k circuits of bottom fan-in $\frac{b}{25ek\log m}$.*

Proof. For the case $2 \leq b \leq k\log m$, since $\frac{b}{25ek\log m} < 1$, the theorem is implied by Theorem 4.3. The case $k\log m < b \leq \sqrt{km\log m/2}$ is proved in Theorem 3.4. For the case $\sqrt{km\log m/2} < b < 2\sqrt{km\log m/2}$, since $\frac{b}{25ek\log m} \leq \frac{\sqrt{km\log m/2}}{12k\log m}$, the theorem is implied by Theorem 3.4. Finally, the case $b \geq 2\sqrt{km\log m/2}$ is proved by Theorem 3.5. □

A number of important consequences follow directly from Theorem 5.1.

THEOREM 5.2. *For any integers $k \geq 1$ and $h \geq 1$, and for any real number r, there are functions that are computable by circuits of linear size and depth k with bottom fan-in $O(\log^h n)$ but that require exponential size for depth k circuits of bottom fan-in $r\log^{h-1} n$.*

Proof. Let $b = 25ek(k-1)^{h-1}r\log^h m$. Note that in this case, the number of variables in the function $f_k^{m,b}$ is $n \leq m^{k-1}$ for m large enough. Thus, $\log m \leq \log n \leq (k-1)\log m$. By the definition, the function $f_k^{m,b}$ can be computed by a depth k and linear size circuit with bottom fan-in $b = O(\log^h n)$. On the other hand, according to Theorem 5.1, the function $f_k^{m,b}$ requires exponential size for depth k circuits whose bottom fan-in is $r(k-1)^{h-1}\log^{h-1} m$. Note that $r(k-1)^{h-1}\log^{h-1} m$ is at least as large as $r\log^{h-1} n$. □

By more careful selections of the bottom fan-in b in Theorem 5.1, combined with a padding technique, we are able to obtain general results for the trade-off between circuit size and circuit bottom fan-in. We illustrate this technique by the following theorem, which can be easily extended to other cases using the same technique.

THEOREM 5.3. *For any integer $k \geq 1$ and for any real number $\epsilon > 0$, there is a function F_k^ϵ that is computable by a circuit of linear size and depth k with bottom fan-in $\log n$ but that requires superpolynomial size for depth k circuits of bottom fan-in $O(\log^{1-\epsilon} n)$.*

Proof. Choose h such that $\frac{h}{h+1} > 1-\epsilon$, and then use Theorem 5.2 to choose a function $f_k^{m,b}$ of $\leq m^{k-1}$ variables which can be computed by a depth k circuit of linear size and bottom fan-in $b = 25ek\log^{h+1} m$ but that requires size

$$2^{\frac{1}{12\sqrt{2k}}\sqrt{\frac{m}{\log m}}} \tag{1}$$

when the bottom fan-in is $\leq \log^h m$.

Now make the function $f_k^{m,b}$ formally the function F_k^ϵ of $n = 2^{25ek\log^{h+1} m}$ variables by adding dummy variables that are not used. The theorem now follows for the function F_k^ϵ since the size bound (1) is superpolynomial in n and $c\log^{1-\epsilon} n < \log^h m$ for any fixed constant c when m is sufficiently large. □

In particular, if we let $\epsilon = 1$ and $h = 1$, then we obtain the following corollary that will be used in section 6.

COROLLARY 5.4. *For any integer $k \geq 1$, there is a function F_k that is computable by a circuit of linear size and depth k with bottom fan-in $\log n$ but that requires superpolynomial size for depth k circuits of bottom fan-in $O(1)$.*

6. Input read-modes of Turing machines. An important application of the above investigation is to the input read-modes of a sublinear-time alternating Turing machine, which is an important computational model in the study of complexity classes.

To make sublinear-time Turing machines meaningful, we allow a Turing machine to have a *random access input tape*, plus a read-write *input address tape*, such that the Turing machine has access to the bit of the input tape denoted by the contents of the input address tape.

An $O(\log n)$-*time alternating Turing machine* (log-*time ATM*) is defined as an extension of the $O(\log n)$-time deterministic Turing machines in the usual way [9]. Given an input, the computation of a log-time ATM M can be represented by an $\wedge$-$\vee$ tree. Each computation path in the $\wedge$-$\vee$ tree can be divided into *phases*, which are the maximal subpaths in which M does not make alternations. The first configuration in each phase is called an *alternation* (configuration). In particular, the starting configuration of M is always an alternation.

A number of input read-modes for sublinear-time alternating Turing machines have appeared in the literature. The standard input read-mode introduced by Chandra, Kozen, and Stockmeyer [9] allows a computation path of an $O(\log n)$-time alternating Turing machine to read up to $\Theta(\log n)$ input bits. Ruzzo [16] proposed an input read-mode in which each computation path can read at most one input bit and the reading must be performed at the end of the path. An input read-mode studied by Sipser [17] insists that the input address tape always be reset to blank after each input reading so that each input reading takes time $\Omega(\log n)$.

It can be shown that many complexity classes such as NC^k for $k \geq 1$ and AC^k for $k \geq 0$ remain the same for all of these input read-modes of alternating Turing machines. On the other hand, it was unknown whether these input read-modes affected the classes of lower complexity such as the levels in the logarithmic time hierarchy. Recently, Cai and Chen [6] have demonstrated how each level of the logarithmic time hierarchy based on each of the above input read-modes can be characterized by a uniform family of circuits. Combining these characterizations with the separation results given in the previous sections, we are able to show that all of these input read-modes are distinct.

Formally, the *logarithmic time hierarchy* is defined as the union of the following classes:

$$\Pi_1, \Pi_2, \ldots, \Pi_k, \ldots,$$

where Π_k is the class of languages accepted by a log-time ATM that always starts with an $\wedge$-state and makes at most k alternations.

The above definition ignores the input read-modes of the log-time ATMs and thus is not very precise. To be more precise, we will call

$$\Pi_1^U, \Pi_2^U, \ldots, \Pi_k^U, \ldots,$$

the logarithmic time hierarchy *based on the Chandra–Kozen–Stockmeyer model*,

$$\Pi_1^R, \Pi_2^R, \ldots, \Pi_k^R, \ldots,$$

the logarithmic time hierarchy *based on Ruzzo's model*, and

$$\Pi_1^S, \Pi_2^S, \ldots, \Pi_k^S, \ldots,$$

the logarithmic time hierarchy *based on Sipser's model*, where Π_k^U (resp., Π_k^R, Π_k^S) is the class of languages accepted by a log-time ATM based on the Chandra–Kozen–Stockmeyer input read-mode (resp., on Ruzzo's input read-mode, on Sipser's input read-mode) that always starts with an $\wedge$-state and makes at most k alternations.

THEOREM 6.1. (See [6].) *For all integers $k \geq 1$,*

(1) *if a language L is in the class Π_k^R, then L is accepted by a Π_k^{poly}-family of circuits;*

(2) *if a language L is in the class Π_k^S, then L is accepted by a $\Pi_k^{poly,c}$-family of circuits for some constant c;*

(3) *if a language L is in the class Π_k^U, then L is accepted by a $\Pi_k^{poly,d\log n}$-family of circuits for some constant d;*

According to the definitions, it is easy to see that $\Pi_k^R \subseteq \Pi_k^S \subseteq \Pi_k^U$. A proof for the inclusion $\Pi_k^U \subseteq \Pi_{k+1}^R$ can be found in [6]. Our main result for this section is that all of these inclusions are strict, as proved in the following theorem.

THEOREM 6.2. *For any integer $k \geq 1$, we have*

$$\Pi_k^R \subset \Pi_k^S \subset \Pi_k^U \subset \Pi_{k+1}^R,$$

where $\subset$ means "proper subset."

Proof. (1) $\Pi_k^R \subset \Pi_k^S$.

It has been proved by Cai and Chen in [6] that $\Pi_1^R \subset \Pi_1^S$. Thus, we only need to prove the strict inclusion for $k \geq 2$.

Without loss of generality, we suppose that the output gate of the tree circuit $C_{k+1}^{m,2}$ defining the function $f_{k+1}^{m,2}$ is an AND gate (otherwise, we consider the negation of the function $f_{k+1}^{m,2}$). Moreover, to make the operations such as m^k, $\log m$, and $\sqrt{m}$ feasible within $O(\log n)$ deterministic time, we consider only the case where m is a power of 2.

Let S_1 be the language whose characteristic function is given by the functions $f_{k+1}^{m,2}$, where m is a power of 2 (in particular, a string is not in the set S_1 if its length does not match the number of variables for any such function $f_{k+1}^{m,2}$). We first construct a log-time ATM M_1 that accepts the set S_1 as follows. On input x, M_1 first computes the length n of x. This can be done in deterministic $O(\log n)$ time by reading $O(\log n)$ input bits [2]. Then M_1 verifies that $n = 2m^{k-1}\sqrt{m/\log m}$ for some integer m that is a power of 2. After this, M_1 simply traces the tree circuit $C_{k+1}^{m,2}$ defining the function $f_{k+1}^{m,2}$, except that in the kth phase, M_1 reads the two consecutive input bits for the corresponding bottom level gate and directly computes the value for the gate. Since the output gate of the circuit $C_{k+1}^{m,2}$ is an AND gate, the log-time ATM M_1 starts with an $\wedge$-state and makes at most k alternations.

According to our assumption, $k \geq 2$. Thus, the log-time ATM M_1 reads at most two input bits in its last phase. By Theorem 3.1 in Cai and Chen [6], M_1 can be simulated by a log-time ATM based on Sipser's input read-mode that always starts with an $\wedge$-state and makes at most k alternations. This proves that the set S_1 is in the class Π_k^S.

Suppose that S_1 is also in the class Π_k^R. Then by Theorem 6.1(1), S_1 is accepted by a Π_k^{poly}-family of circuits. Thus, for any integer m that is a power of 2, the function $f_{k+1}^{m,2}$ is computable by a depth k circuit of polynomial size. This contradicts Corollary 4.4.

This contradiction shows $\Pi_k^R \subset \Pi_k^S$.

(2) $\Pi_k^S \subset \Pi_k^U$.

The proof is similar to that for case (1). Consider the function F_{k+1} in Corollary 5.4, which is a function of n variables obtained from the function $f_{k+1}^{m,b}$, $b = 25e(k+1)\log^2 m$ by adding dummy variables, where $n = 2^{25e(k+1)\log^2 m}$. We also make similar assumptions as we did for case (1). Thus, the output gate of the tree circuit $C_{k+1}^{m,b}$ defining the function $f_{k+1}^{m,b}$ is an AND gate, and m is a power of 2. Under these assumptions, it is easy to see that the set S_2 whose characteristic function is given by the functions F_{k+1} in Corollary 5.4 is in the class Π_k^U: a log-time ATM M_2 first verifies the length of the input and traces the tree circuit $C_{k+1}^{m,b}$ defining the corresponding function $f_{k+1}^{m,b}$ except that in the kth phase, M_2 reads directly a consecutive block of b input bits that are the inputs to the corresponding bottom level gate. Note that the b consecutive input bits can be read in deterministic $O(b + \log n)$ time [4], and that b is logarithmic in the input length n of the function F_{k+1}. This proves that the language S_2 is in the class Π_k^U.

Suppose that S_2 is also in the class Π_k^S. By Theorem 6.1(2), the language S_2 is accepted by a $\Pi_k^{poly,c}$-family of circuits for some constant c. That is, the function F_{k+1} is computable by a depth $k+1$ and bottom fan-in c circuit whose size is polynomial. But this contradicts Corollary 5.4. Thus, $\Pi_k^S \subset \Pi_k^U$.

(3) $\Pi_k^U \subset \Pi_{k+1}^R$.

The proof is similar to those of the other two cases. Let S_3 be the language whose characteristic function is given by the Sipser functions f_{k+1}^m. Then the set S_3 can be accepted by a log-time ATM M_3 that always starts with an $\wedge$-state and makes at most $k+1$ alternations. Moreover, the last phase of M_3 reads at most one input bit. By Theorem 3.1 in Cai and Chen [6], M_3 can be simulated by a log-time ATM based on Ruzzo's input read-mode that always starts with an $\wedge$-state and makes at most $k+1$ alternations. Thus, the set S_3 is in the class Π_{k+1}^R. On the other hand, by Theorem 6.1(3), $S_3 \in \Pi_k^U$ would imply that S_3 is accepted by a $\Pi_k^{poly,O(\log n)}$-family of circuits. That would in turn imply that the Sipser function f_{k+1}^m is computable by a depth $k+1$ circuit of polynomial size whose bottom fan-in is $O(\log n)$, contradicting Theorem 2.2.

This completes the proof. □

COROLLARY 6.3. *For each $k \geq 1$, the kth levels of the logarithmic time hierarchy based on the Chandra–Kozen–Stockmeyer input read-mode, Sipser's input read-mode, and Ruzzo's input read-mode are all distinct.*

Acknowledgments. The second author would like to thank Mike Sipser for an early discussion that initialized this line of research. He is also grateful to Ken Regan and Rong Chen for their comments and constructive discussions. Finally, the authors are especially thankful to two anonymous referees for comments and suggestions that have improved the presentation. In particular, one of the referees pointed out a technical bug in an earlier version of the present paper.

REFERENCES

[1] M. AJTAI, *Σ_1^1-formulae on finite structures*, Ann. Pure Appl. Logic, 24 (1983), pp. 1–48.

[2] D. Barrington, N. Immerman, and H. Straubing, *On uniformity within NC^1*, J. Comput. System Sci., 41 (1990), pp. 274–306.
[3] R. B. Boppana and M. Sipser, *The complexity of finite functions*, in Handbook of Theoretical Computer Science Vol. A, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 757–804.
[4] S. R. Buss, *The Boolean formula value problem is in ALOGTIME*, in Proc. 19th Annual ACM Symposium on Theory of Computing, ACM, New York, 1987, pp. 123–131.
[5] L. Cai and J. Chen, *Fixed parameter tractability and approximability of NP-hard optimization problems*, J. Comput. System Sci., 54 (1997), pp. 465–474.
[6] L. Cai and J. Chen, *On input read-modes of alternating Turing machines*, Theoret. Comput. Sci., 148 (1995), pp. 33–55.
[7] L. Cai and J. Chen, *On the amount of nondeterminism and the power of verifying*, SIAM J. Comput., 26 (1997), pp. 733–750.
[8] L. Cai, J. Chen, R. G. Downey, and M. R. Fellows, *On the structure of parameterized problems in NP*, Inform. and Comput., 123 (1995), pp. 38–49.
[9] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer, *Alternation*, J. Assoc. Comput. Mach., 28 (1981), pp. 114–133.
[10] J. Chen, *Characterizing parallel hierarchies by reducibilities*, Inform. Process. Lett., 39 (1991), pp. 303–307.
[11] S. Cook, *A taxonomy of problems with fast parallel algorithms*, Inform. and Control, 64 (1985), pp. 2–22.
[12] M. Furst, B. Saxe, and M. Sipser, *Parity, circuits, and the polynomial-time hierarchy*, Math. Systems Theory, 17 (1984), pp. 13–27.
[13] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, Addison-Wesley, Reading, MA, 1989.
[14] J. Håstad, *Computational Limitations for Small-Depth Circuits*, The MIT Press, Cambridge, MA, 1986.
[15] J. Håstad, *Almost optimal lower bounds for small depth circuits*, in Advances in Computing Research 5, S. Micali, ed., JAI Press Inc., Greenwich, CT, 1989, pp. 143–170.
[16] W. L. Ruzzo, *On uniform circuit complexity*, J. Comput. System Sci., 22 (1981), pp. 365–383.
[17] M. Sipser, *Borel sets and circuit complexity*, in Proc. 15th Annual ACM Symposium on Theory of Computing, ACM, New York, 1983, pp. 61–69.
[18] A. C. Yao, *Separating the polynomial-time hierarchy by oracles*, in Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1985, pp. 1–10.

SIAM J. COMPUT.
Vol. 27, No. 2, pp. 356–400, April 1998
© 1998 Society for Industrial and Applied Mathematics
003

ON THE COMPLEXITY OF COMPUTING MIXED VOLUMES*

MARTIN DYER†, PETER GRITZMANN‡, AND ALEXANDER HUFNAGEL§

Abstract. This paper gives various (positive and negative) results on the complexity of the problem of computing and approximating mixed volumes of polytopes and more general convex bodies in arbitrary dimension.

On the negative side, we present several #$\mathbb{P}$-hardness results that focus on the difference of computing mixed volumes versus computing the volume of polytopes. We show that computing the volume of zonotopes is #$\mathbb{P}$-hard (while each corresponding mixed volume can be computed easily) but also give examples showing that computing mixed volumes is hard even when computing the volume is easy.

On the positive side, we derive a randomized algorithm for computing the mixed volumes

$$V(\overbrace{K_1,\dots,K_1}^{m_1},\overbrace{K_2,\dots,K_2}^{m_2},\dots,\overbrace{K_s,\dots,K_s}^{m_s})$$

of well-presented convex bodies $K_1,\dots,K_s$, where $m_1,\dots,m_s \in \mathbb{N}_0$ and $m_1 \geq n-\psi(n)$ with $\psi(n)=o(\frac{\log n}{\log\log n})$. The algorithm is an interpolation method based on polynomial-time randomized algorithms for computing the volume of convex bodies.

This paper concludes with applications of our results to various problems in discrete mathematics, combinatorics, computational convexity, algebraic geometry, geometry of numbers, and operations research.

Key words. computational convexity, volume, mixed volumes, convex body, polytope, zonotope, parallelotope, computation, approximation, computational complexity, deterministic algorithm, randomized algorithm, polynomial-time algorithm, $\mathbb{NP}$-hardness, #$\mathbb{P}$-hardness, permanent, determinant problems, lattice point enumerator, partial order, Newton polytope, polynomial equations

AMS subject classifications. 52B55, 52A39, 68Q20, 68Q15, 68R05, 68U05, 52A20, 90C30, 90C25

PII. S0097539794278384

Introduction. The present paper deals with algorithmic questions related to the problem of computing or approximating volumes and mixed volumes of convex bodies by means of deterministic or randomized algorithms. The emphasis will be on the case of varying dimension (but we will also mention some results for fixed dimension).

As the terms are used here, a *convex body* in $\mathbb{R}^n$ is a nonempty compact convex set and a *polytope* is a convex body that has only finitely many extreme points. A convex body or a polytope in $\mathbb{R}^n$ is called *proper* if it is n-dimensional and hence has nonempty interior. A convenient way to deal algorithmically with general convex bodies is to assume that the convex body in question is "well presented" by an algorithm (called an *oracle*) that answers certain sorts of questions about the body and also gives some a priori information; see subsection 1.2 for precise definitions.

The problem of computing the volume $\mathrm{vol}_n(K)$ of an appropriately presented convex body K of $\mathbb{R}^n$ is of fundamental importance from both a theoretical and computational point of view. If K is of the form $K=\sum_{i=1}^s \lambda_i K_i$, where $K_1,\dots,K_s$

*Received by the editors December 9, 1994; accepted for publication (in revised form) January 23, 1996. Research of each author was supported in part by the Deutsche Forschungsgemeinschaft. Research of P. Gritzmann was supported in part by a Max-Planck Research Award.

http://www.siam.org/journals/sicomp/27-2/27838.html

†School of Computer Studies, University of Leeds, Leeds LS2 9JT, U.K. (dyer@dcs.leeds.ac.uk).

‡University of Technology Munich, Center for Mathematical Sciences, D-80290 Munich, Germany (gritzman@mathematik.tu-muenchen.de).

§Vogelherd 9, 90542 Eckental, Germany.

are convex bodies and $\lambda_1, \ldots, \lambda_s$ are positive reals, then $\mathrm{vol}_n(K)$ can be expressed in terms of the *mixed volumes* $V(K_{i_1}, K_{i_2}, \ldots, K_{i_n})$ of $K_1, \ldots, K_s$; in fact,

$$V\left(\sum_{i=1}^{s} \lambda_i K_i\right) = \sum_{i_1=1}^{s} \sum_{i_2=1}^{s} \cdots \sum_{i_n=1}^{s} \lambda_{i_1} \lambda_{i_2} \cdots \lambda_{i_n} V(K_{i_1}, K_{i_2}, \ldots, K_{i_n})$$

is a multivariate homogeneous polynomial of degree n in the variables $\lambda_1, \ldots, \lambda_s$; see subsection 1.1.

The corresponding *Brunn–Minkowski* theory is the backbone of convexity theory (see [Sc93]), but it is also relevant for numerous applications in combinatorics, algebraic geometry and a number of other areas; see section 4 and [GK94].

As it is well known, $V(K, \ldots, K) = \mathrm{vol}_n(K)$, and hence, mixed volumes generalize the ordinary volume. From this observation it is already clear that, in general, any hardness result for volume computation carries over to mixed volumes. Specifically, the problem of computing the volume of polytopes (given in terms of their vertices—"$\mathcal{V}$-polytopes"—or in terms of their facet hyperplanes—"$\mathcal{H}$-polytopes"; see subsection 1.2) is known to be #$\mathbb{P}$-hard (see [DF88]), whence computing mixed volumes of polytopes is also (at least) #$\mathbb{P}$-hard.

The hardness issue of volume versus mixed volume computation is, however, more complicated than that. In the case where the number of bodies is not bounded beforehand but part of the input, the above multivariate polynomial typically has exponentially many coefficients and this implies that the task of computing *all* mixed volumes of a given set of bodies does require exponential time. But this fact also allows for the possibility that the volume of the Minkowski sum of convex bodies may be hard to compute even if each mixed volume can be computed easily. Indeed, when the bodies $K_1, \ldots, K_s$ are all line segments, each mixed volume computation is just the evaluation of a corresponding determinant; computing the volume of the zonotope $K = K_1 + \cdots + K_s$ is, in general however, hard.

THEOREM 1. *The following task is #$\mathbb{P}$-hard: given $n, s \in \mathbb{N}$ and rational vectors $z_1, \ldots, z_s$ of $\mathbb{R}^n$, compute the volume of the zonotope $\sum_{i=1}^{s} [0,1] z_i$.*

A slight strengthening of this result is contained in Theorem 5. As a corollary to Theorem 1 we show in Theorem 2 that (approximately) computing the volume of the Minkowski sum of ellipsoids is also #$\mathbb{P}$-hard, a result needed in subsection 2.4.

Conversely to Theorem 1, computing *a single* mixed volume may be hard even if the volume of the corresponding Minkowski sum is easy to compute.

THEOREM 3. *The following problem MIXED-VOLUME-OF-BOXES is #$\mathbb{P}$-hard: given a positive integer n and, for $i, j = 1, 2, \ldots, n$, positive rationals $\alpha_{i,j}$, determine the mixed volume $V(Z_1, \ldots, Z_n)$ of the axes-parallel parallelotopes $Z_i = \sum_{j=1}^{n} [0, \alpha_{i,j}] e_j$, $(i = 1, 2, , \ldots, n)$, where e_j denotes the jth unit vector.*

The #$\mathbb{P}$-hardness persists even when the boxes are restricted to having just two different (and previously prescribed) edge lengths.

Proofs of these theorems (and related results) are given in section 2. We further show that Theorem 3 can be strengthened to just two parallelotopes if one of them is permitted to deviate from being axes-parallel (Theorem 4). In view of these results it may be surprising that even though the computation of certain mixed volumes appears to be harder than volume computation, from the point of view of complexity theory it is not. Theorems 6 and 7 show that the problem of computing any specific mixed volume of polytopes (or zonotopes) is #$\mathbb{P}$-easy.

Section 2 will also discuss the problem of how efficiently mixed volumes can be approximated by means of deterministic algorithms. [GLS88], [AK90], and [BH93]

give exponential upper bounds for the error of deterministic polynomial approximations of the volume, and [BF86] gives an almost matching lower bound in the oracular model. We discuss possible extensions to mixed volumes and derive a polynomial-time algorithm for estimating any mixed volume of two convex bodies to a relative error that depends only on the dimension but is independent of the "well-boundedness" parameters of the bodies (Theorem 9). As a necessary "by-product" we further show that it can be decided in polynomial time whether the mixed volume of convex bodies vanishes (Theorem 8). This is a nontrivial result since a mixed volume may be greater than zero even if each set is contained in a lower-dimensional affine subspace.

A natural approach to mixed volumes is to try to use values (or estimates thereof) of the polynomial $\mathrm{vol}_n(\sum_{i=1}^s \lambda_i K_i)$ for computing (or estimating) (some of) its coefficients, the mixed volumes of the convex bodies K_i. This approach works under reasonable assumptions provided the above polynomial can be evaluated (approximately) in polynomial time; see [GK94]. This is particularly true for polytopes in *fixed* dimension; see [AS86], [CH79]. For variable dimension there is not much hope in ever obtaining a polynomial-time deterministic algorithm for this task, but we may utilize the polynomial-time randomized volume algorithm of [DFK91].

PROPOSITION 1. *There is a polynomial-time randomized algorithm which solves the following problem:*

Instance: *A well-presented convex body K in $\mathbb{R}^n$, positive rational numbers τ and β.*

Output: *A random variable $\hat{v} \in \mathbb{Q}$ such that*

$$prob\left\{\frac{|\hat{v} - \mathrm{vol}_n(K)|}{\mathrm{vol}_n(K)} \geq \tau\right\} \leq \beta.$$

Let us point out that after a preprocessing "rounding" step whose running time depends on the "a priori parameters" of the body, the running time of the main algorithm is bounded above by a polynomial in n, $\frac{1}{\tau}$, and $\log(\frac{1}{\beta})$.

[DFK91]'s algorithm was improved by [LS90], [AK90], [DF91], [LS93], [KLS97]; see [Kh93], [GK94], and [Lo95] for surveys. Let us point out that, when dealing with randomized algorithms of the above kind, it suffices to give the desired approximation to error probability, say $\frac{1}{4}$. Then after $O(\log(1/\beta))$ independent trials of the algorithm, the median of the results achieves the required probability β; see [JVV86], [SJ89], [KKLLL93], or [LS93].

Even for just two bodies there are two major difficulties in extending Proposition 1 to mixed volumes. First, in general there is *no* way of obtaining *relative* estimates of the coefficients from *relative* estimates of the values of a polynomial p. (This is easily seen by considering the one-parameter sequence of univariate polynomials $q_\beta(x) = 1 + \beta x + x^2$, where β may be any arbitrary small positive rational number; cf. [GK94, section 6.2]). The special structure of the "mixed volume polynomial" $p(x) = \mathrm{vol}_n(K_1 + xK_2)$ will, however, allow us to handle this problem. Second, the absolute values of the entries of the "inversion" which is used for expressing the coefficients of the polynomial in terms of its approximated values are not bounded by a polynomial, while the randomized volume approximation algorithm is polynomial only in $\frac{1}{\tau}$ but not in $size(\tau)$. This difficulty is mirrored in the restrictions on ψ in the following theorem.

THEOREM 10. *Suppose that $\psi : \mathbb{N} \to \mathbb{N}$ is nondecreasing with*

$$\psi(n) \leq n \qquad \text{and} \qquad \psi(n)\log\psi(n) = o(\log n).$$

Then there is a polynomial-time algorithm for the following problem:

Instance: *Well-presented convex bodies K_1, K_2 of $\mathbb{R}^n$, positive rational numbers ϵ and β, an integer m with $0 \le m \le \psi(n)$.*

Output: *The information that the mixed volume*

$$a_m = V(\overbrace{K_1, \dots, K_1}^{n-m}, \overbrace{K_2, \dots, K_2}^{m})$$

of K_1 and K_2 vanishes, iff $a_m = 0$, or, otherwise, a random variable $\hat{a}_m \in \mathbb{Q}$, satisfying

$$prob\left\{\frac{|\hat{a}_m - a_m|}{a_m} \ge \epsilon\right\} \le \beta.$$

The complexity of the above algorithm is only marginally worse than the complexity of the volume oracle; see section 3 for a detailed analysis. Note that the function

$$\psi(n) = \left\lceil \frac{\log n}{\log^2 \log n} \right\rceil$$

satisfies the above condition (on $\mathbb{N} \setminus \{1, 2, 3\}$).

Theorem 10 can be extended to more than two bodies.

THEOREM 11. *Suppose that $\psi : \mathbb{N} \to \mathbb{N}$ is nondecreasing with*

$$\psi(n) \le n \qquad \text{and} \qquad \psi(n) \log \psi(n) = o(\log n).$$

Then there is a polynomial-time algorithm for the following problem:

Instance: *$n, s \in \mathbb{N}$, $m_1, \dots, m_s \in \mathbb{N}_0$ with $m_1 + m_2 + \cdots + m_s = n$ and $m_1 \ge n - \psi(n)$, well-presented convex bodies $K_1, \dots, K_s$ of $\mathbb{R}^n$, positive rational numbers ϵ and β.*

Output: *The information that the mixed volume*

$$V_{m_1,\dots,m_s} = V(\overbrace{K_1, \dots, K_1}^{m_1}, \dots, \overbrace{K_s, \dots, K_s}^{m_s})$$

vanishes, iff $V_{m_1,\dots,m_s} = 0$, or, otherwise, a random variable $\hat{V}_{m_1,\dots,m_s} \in \mathbb{Q}$ such that

$$prob\left\{\frac{|\hat{V}_{m_1,\dots,m_s} - V_{m_1,\dots,m_s}|}{V_{m_1,\dots,m_s}} \ge \epsilon\right\} \le \beta.$$

Theorems 10 and 11 will be proved in section 3. But let us take a few words here to place their results into perspective. Both theorems are proved by using an interpolation (or numerical differentiation) method, which is based on Proposition 1. A special feature of such a method is that in order to compute *a specific* coefficient of the polynomial under consideration it computes essentially *all* (or at least "all previous") coefficients. Now, suppose that $\psi : \mathbb{N} \to \mathbb{N}$ is a functional with $\psi(n) \le n$ for all $n \in \mathbb{N}$; let

$$\mathcal{I}_\psi(n) = \{(m_1, \dots, m_{\psi(n)}) : m_1, \dots, m_{\psi(n)} \in \mathbb{N}_0,\ m_1 + \cdots + m_{\psi(n)} = n \\ \text{and } n - m_1 \le \psi(n)\},$$

and let $K_1, \dots, K_{\psi(n)}$ be convex bodies of $\mathbb{R}^n$. Then

$$|\mathcal{I}_\psi(n)| = \binom{2\psi(n)-1}{\psi(n)-1},$$

whence the number of different mixed volumes

$$V(\overbrace{K_1, \dots, K_1}^{m_1}, \overbrace{K_2, \dots, K_2}^{m_2}, \dots, \overbrace{K_{\psi(n)}, \dots, K_{\psi(n)}}^{m_{\psi(n)}})$$

is in general only bounded by a polynomial in n if $\psi(n) \le \kappa \log n$ for some constant κ. This means that there can possibly be a polynomial-time algorithm for computing *all such* mixed volumes only if

$$\psi(n) \le \kappa \log n.$$

As we will see in section 3, the statements of Theorems 10 and 11 are much easier to prove for ψ being constant. As the previous discussion shows, when the number of bodies is part of the input, no polynomial-time algorithm is capable of computing more than "very few" mixed volumes. This fact places severe limitations on interpolation methods that indicate that the restriction on ψ in Theorem 11 is "essentially best-possible" for any such method.

Let us remark that, for general convex bodies, it is an open problem whether there exists any method that avoids these limitations and allows one to access *single specific* mixed volumes. Hence it is open, whether the above restrictions on ψ can be lifted and whether there are polynomial-time randomized algorithms which, on arbitrarily *given* $n, s \in \mathbb{N}$, $m_1, \dots, m_s \in \mathbb{N}_0$ *with* $m_1 + m_2 + \cdots + m_s = n$, *well-presented convex bodies* $K_1, \dots, K_s$ *of* $\mathbb{R}^n$ *and positive rational numbers* ϵ *and* β, *compute a random variable* $\hat{V}_{m_1,\dots,m_s} \in \mathbb{Q}$ *such that* $\text{prob}\{|\hat{V}_{m_1,\dots,m_s} - V_{m_1,\dots,m_s}|/V_{m_1,\dots,m_s} \ge \epsilon\} \le \beta$.

Note specifically that even the case $s = n$, $m_1 = \cdots = m_s = 1$ is open.

Section 4 contains some problems related to mixed volumes and some applications of our results. In particular, we deal with the problem of counting the number of integer points in lattice polytopes and with some determinant problems involving minors of given matrices. Furthermore, we discuss possible applications of our results to problems in mixture management, combinatorics, and algebraic geometry.

1. Basic geometric and computational aspects. The following three subsections provide definitions, notation, background information, and some first results that are needed later in sections 2 and 3.

1.1. Mixed volumes. Let $\mathcal{K}^n$ denote the family of all convex bodies of $\mathbb{R}^n$.

A theorem of Minkowski [Mi11] (see also [BF34], [Sc93, section 5]) shows that for $K_1, K_2, \dots, K_s \in \mathcal{K}^n$ and nonnegative reals $\lambda_1, \lambda_2, \dots, \lambda_s$,

$$\text{vol}_n\left(\sum_{i=1}^{s} \lambda_i K_i\right)$$

is a homogeneous polynomial of degree n in $\lambda_1, \dots, \lambda_s$, and can be written in the form

$$\text{vol}_n\left(\sum_{i=1}^{s} \lambda_i K_i\right) = \sum_{i_1=1}^{s} \sum_{i_2=1}^{s} \cdots \sum_{i_n=1}^{s} \lambda_{i_1}\lambda_{i_2}\cdots\lambda_{i_n} V(K_{i_1}, K_{i_2}, \dots, K_{i_n}), \tag{1.1}$$

where the coefficients $V(K_{i_1}, K_{i_2}, \ldots, K_{i_n})$ are order-independent, i.e., invariant under permutations of their arguments. The coefficient $V(K_{i_1}, K_{i_2}, \ldots, K_{i_n})$ is called the *mixed volume* of $K_{i_1}, K_{i_2}, \ldots, K_{i_n}$. We will also use the term *mixed volume* for the functional

$$V : \overbrace{\mathcal{K}^n \times \cdots \times \mathcal{K}^n}^{n} \to \mathbb{R}, \qquad (K_1, \ldots, K_n) \mapsto V(K_1, \ldots, K_n),$$

as well as for restrictions of this functional to certain subsets of $\mathcal{K}^n \times \cdots \times \mathcal{K}^n$. Mixed volumes are nonnegative, monotone, multilinear, and continuous valuations; see [BZ88, Chapter 4], [Sa93], and [GK94] for the basic properties of mixed volumes, and see [Sc93] for an excellent detailed treatment of the Brunn-Minkowski theory.

The order-independence gives rise to the notation

$$V(\overbrace{K_1, \ldots, K_1}^{m_1}, \overbrace{K_2, \ldots, K_2}^{m_2}, \ldots, \overbrace{K_s, \ldots, K_s}^{m_s})$$

for the mixed volume $V(K_1, \ldots, K_s)$, where each K_i occurs exactly m_i times and $\sum_{i=1}^{s} m_i = n$. The following *Aleksandrov–Fenchel inequality*, [Al37], [Al38], [Fe36], plays a fundamental role in the Brunn-Minkowski theory and will be needed in the approximation algorithm of section 3.

$$V(K_1, K_2, K_3, \ldots, K_n)^2 \geq V(K_1, K_1, K_3, , \ldots, K_n)\ V(K_2, K_2, K_3, \ldots, K_n), \tag{1.2}$$

whenever $K_1, K_2, \ldots, K_n \in \mathcal{K}^n$; see [Sc93] for a proof and a discussion of this inequality.

The following "decomposition lemma" (see, e.g., [BZ88, section 19.4]) will also turn out to be useful in our analysis.

PROPOSITION 2. *Let $K_1, \ldots, K_n \in \mathcal{K}^n$ and suppose that $K_{n-m+1}, \ldots, K_n$ are contained in some m-dimensional affine subspace U of $\mathbb{R}^n$. Let V_U denote the mixed volume with respect to the m-dimensional volume measure on U, and let $V_{U^\perp}$ be defined similarly with respect to the orthogonal complement $U^\perp$ of U. Then*

$$\binom{n}{m} V(K_1, \ldots, K_{n-m}, K_{n-m+1}, \ldots, K_n) = V_{U^\perp}(K_1', \ldots, K_{n-m}')V_U(K_{n-m+1}, \ldots, K_m),$$

where $K_1', \ldots, K_{n-m}'$ denote the orthogonal projections of $K_1, \ldots, K_{n-m}$ onto $U^\perp$, respectively.

As a particular consequence, it follows that

$$V(K_1, \ldots, K_{n-m}, K_{n-m+1}, \ldots, K_n) = 0$$

if there is a proper subspace of U that contains $K_{n-m+1}, \ldots, K_n$. (Note, however, that in general the mixed volume may be greater than zero even if each set lies in *some* lower-dimensional subspace of $\mathbb{R}^n$.) In the special case $m = 1$, $K_n = [0, 1]v$, and $U = \mathrm{lin}\{v\}$, where $v \in \mathbb{R}^n \setminus \{0\}$, Proposition 2 reads

$$n \cdot V(K_1, \ldots, K_{n-1}, [0, 1]v) = \|v\| \cdot V_{U^\perp}(K_1', \ldots, K_{n-1}').$$

If all bodies $K_1, \ldots, K_s$ are line segments, say

$$K_i = S_i = p_i + [0, 1]z_i \quad (i = 1, \ldots, s),$$

with $p_i, z_i \in \mathbb{R}^n$, then $Z = \sum_{i=1}^{s} S_i$ is a *zonotope*. It follows that for any sequence $1 \le i_1, i_2, \dots, i_n \le s$ of mutually distinct indices,

$$V(S_{i_1}, S_{i_2}, \dots, S_{i_n}) = \frac{1}{n!}\big|\det(z_{i_1}, z_{i_2}, \dots, z_{i_n})\big|,$$

where $(z_{i_1}, z_{i_2}, \dots, z_{i_n})$ denotes the $n \times n$-matrix with columns $z_{i_1}, \dots, z_{i_n}$. With the aid of (1.1) this implies the well-known volume formula for zonotopes,

$$\mathrm{vol}_n\left(\sum_{i=1}^{s} S_i\right) = \sum_{1 \le i_1 < i_2 < \cdots < i_n \le s} \big|\det(z_{i_1}, z_{i_2}, \dots, z_{i_n})\big|; \tag{1.3}$$

see [Sh74], [Mo89], or [St91].

The polynomial expression in (1.1) involves all s variables $\lambda_1, \dots, \lambda_s$ but, of course, one of the variables, say λ_1, may be set to 1, whence the problem of computing all mixed volumes of s sets in $\mathbb{R}^n$ can be reduced to the task of computing the coefficients of a (generally now inhomogenous) polynomial of degree n in $s-1$ indeterminates. For $s = 2$ we obtain the univariate polynomial

$$p(x) = \mathrm{vol}_n(K_1 + xK_2) = \sum_{i=0}^{n} \binom{n}{i} a_i x^i,$$

where

$$a_i = V(\overbrace{K_1, \dots, K_1}^{n-i}, \overbrace{K_2, \dots, K_2}^{i}).$$

The *Aleksandrov–Fenchel inequality* implies that the sequence

$$q_m = \frac{a_{m-1}}{a_m} \qquad (m = 1, \dots, n)$$

is increasing, whence the sequence $a_0, \dots, a_n$ is *unimodal*.

Finally, note that when $s > 2$, $m_1, m_2, \dots, m_s \in \mathbb{N}_0$ with $\sum_{i=1}^{s} m_i = n$, and $L_x = K_0 + xK_1$ for nonnegative $x \in \mathbb{R}$,

$$q(x) = V(\overbrace{L_x, \dots, L_x}^{m_1}, \overbrace{K_2, \dots, K_2}^{m_2}, \dots, \overbrace{K_s, \dots, K_s}^{m_s})$$

is a polynomial in x of degree m_1, and we have

$$q(x) = \sum_{k=0}^{m_1} \binom{m_1}{k} V(\overbrace{K_0, \dots, K_0}^{m_1-k}, \overbrace{K_1, \dots, K_1}^{k}, \overbrace{K_2, \dots, K_2}^{m_2}, \dots, \overbrace{K_s, \dots, K_s}^{m_s}) x^k. \tag{1.4}$$

This fact will, in particular, be used in subsection 3.3.

1.2. Algorithmic preliminaries. The present subsection begins with some remarks on how to deal algorithmically with polytopes and more general convex bodies, and then collects a few results that are needed later.

The underlying model of computation is the binary Turing machine model, which—in case of convex bodies—will be augmented by certain oracles; see [GJ79], [GLS88].

From an algorithmic point of view, polytopes are dealt with much more easily than general convex bodies because polytopes can be presented in a finite manner, namely, in terms of their vertices or in terms of their facet halfspaces. Clearly, from an algorithmic point of view it is not the geometric object that is relevant but its presentation. Hence we use the following notation; see e.g., [GK94].

A string $(n, m; v_1, \ldots, v_m)$ with $n, m \in \mathbb{N}$, and $v_1, \ldots, v_m \in \mathbb{Q}^n$ is called a $\mathcal{V}$-*polytope* in $\mathbb{R}^n$; it represents the geometric object $P = \operatorname{conv}\{v_1, \ldots, v_m\}$; hence we will sometimes write $P = (n, m; v_1, \ldots, v_m)$. A string $(n, m; A, b)$, where $n, m \in \mathbb{N}$, A is a rational $m \times n$ matrix and $b \in \mathbb{Q}^m$ such that $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ is a polytope is called an $\mathcal{H}$-*polytope* in $\mathbb{R}^n$, and is again identified with the geometric object P. If we want to focus more on the geometric object P we will call each corresponding $\mathcal{V}$- or $\mathcal{H}$-polytope a $\mathcal{V}$- or $\mathcal{H}$-*presentation* of P. The *binary size* (or short *size*) of a $\mathcal{V}$- or $\mathcal{H}$-polytope P is the number of binary digits needed to encode the data of the presentation.

Let us point out that each rational polytope admits a presentation as a $\mathcal{V}$- or $\mathcal{H}$-polytope, and in fixed dimension one can be computed from the other in polynomial time. This is no longer true in general when the dimension is part of the input, since the number of vertices of a polytope may be exponential in its number of facets and vice versa; see [Mc70].

Zonotopes admit specifically "compact" presentations. A string $(n, s; c; z_1, \ldots, z_s)$ with $n, s \in \mathbb{N}$ and $c, z_1, \ldots, z_s \in \mathbb{Q}^n$ is called an $\mathcal{S}$-*zonotope* in $\mathbb{R}^n$; it represents the geometric object $Z = c + \sum_{i=1}^{s} [0, 1] z_i$. Sometimes we will also work with zonotopes whose relationship to the origin (and whose scaling) is different. Specifically, zontopes of the form $\sum_{i=1}^{s} [-1, 1] z_i$ will be used. To keep the notation simple, we refrain, however, from introducing an additional name for such a presentation. Note that, in general, neither the vertices nor the facets of a zonotope are readily accessible from an $\mathcal{S}$-presentation. In fact, for zonotopes generated by s segments in general position, both the number of facets and the number of vertices grow exponentially as m increases.

A zonotope Z is called a *parallelotope* if the "generators" $z_1, \ldots, z_s$ are linearly independent; it is *rectangular* if they are pairwise othogonal, and *axes-parallel* if all generators are standard unit vectors.

A convenient way to deal algorithmically with general convex bodies K is to assume that K is given by an algorithm (called an *oracle*) that answers certain sorts of questions about the body. These oracles are designed in such a way that the standard polytope case is included, i.e., it is easy to construct the corresponding oracles for $\mathcal{V}$- or $\mathcal{H}$-polytopes. This oracular approach has been introduced and extensively studied for proper convex bodies in [GLS88]. In particular, [GLS88] shows that under suitable additional assumptions, "membership," "separation," and "optimization" are equivalent. Here we need a slight variant since we want to deal with mixed volumes of possibly improper convex bodies. Let $K \in \mathcal{K}^n$, and define for $\epsilon \geq 0$ the *outer parallel body* and the *inner parallel body* of K, respectively, by

$$K(\epsilon) = (K + \epsilon \mathbb{B}^n) \cap \operatorname{aff}(K) \quad \text{and} \quad K(-\epsilon) = K \setminus ((\operatorname{aff}(K) \setminus K) + \epsilon \mathbb{B}^n),$$

where $\mathbb{B}^n$ denotes the Euclidean unit ball in $\mathbb{R}^n$. The most natural algorithmic problem for convex bodies K is the following.

WEAK MEMBERSHIP PROBLEM FOR $K \in \mathcal{K}^n$. *Given $y \in \mathbb{Q}^n$, and a rational number $\epsilon > 0$, assert that $y \in K(\epsilon)$ or that $y \notin K(-\epsilon)$.*

If a convex body K is given by an algorithm that solves the weak membership problem, we say that K is described by a *weak membership oracle*. It is quite evi-

dent that the information given by a weak membership oracle is insufficient for most algorithmic purposes. Hence we need some additional a priori information about the body in question; see [GLS88] for a discussion of these assumptions in case of a proper convex body. A convex body K of $\mathbb{R}^n$ will be called *well presented* if it is given by a weak membership oracle and if the following additional information is provided: a nonnegative integer d and vectors $a_0, \dots, a_d \in \mathbb{Q}^n$ such that $\operatorname{aff}(K) = \operatorname{aff}\{a_0, \dots, a_d\}$; a vector $b \in Kn\mathbb{Q}^n$, and positive rational numbers ρ and R such that $(b + \rho\mathbb{B}^n) \cap \operatorname{aff}(K) \subset K \subset R\mathbb{B}^n$.

Note that d is the dimension of K and that $\operatorname{aff}(K)$ is presented in terms of an affine basis. It is, however, easy to compute from $a_0, \dots, a_d$ a presentation of $\operatorname{aff}(K)$ as the solution space of a system of (rational) linear equations and vice versa. The *size* of a well-presented convex body $K \in \mathcal{K}^n$ is then defined as n plus the sum of the binary sizes of the parameters $a_0, \dots, a_d$, b, ρ, and R, and the *input size* of the weak membership oracle for K is the sum of $size(K)$ and $size(\epsilon)$.

It is not hard to see that well presentation carries over to Minkowski sums. In fact, let $K_1, \dots, K_s \in \mathcal{K}^n$ be well presented with parameters n, d_i, $a_{i,0}, \dots, a_{i,d_i}$, b_i, ρ_i, and R_i for $i = 1, \dots, s$, and let $\lambda_1, \dots, \lambda_s$ be positive rationals whose sizes are bounded above by the sizes of $K_1, \dots, K_s$. Then it is easy to find an affine basis of the affine hull of $K = \lambda_1 K_1 + \cdots + \lambda_s K_s$, and

$$
\begin{aligned}
&d = \dim(\operatorname{lin}\{a_{i,j} - a_{i,0} : i = 1, \dots, s; j = 1, \dots, d_i\}),\\
&b = \lambda_1 b_1 + \cdots + \lambda_s b_s,\\
&R = \lambda_1 R_1 + \cdots + \lambda_s R_s
\end{aligned}
$$

are valid parameters for K. Further, one can compute in polynomial time a nontrivial lower bound ρ such that $(b+\rho\mathbb{B}^n)\cap\operatorname{aff}(K) \subset K$. It is also true that a membership oracle for K can be derived in polynomial time from membership oracles for $K_1, \dots, K_s$, but this result makes use of the nontrivial relation of the oracles studied in [GLS88].

PROPOSITION 3. *Let $K_1, \dots, K_s \in \mathcal{K}^n$ be well presented, and let $\lambda_1, \dots, \lambda_s$ be positive rationals whose sizes are bounded above by the sizes of $K_1, \dots, K_s$. Then, a well presentation for $K = \lambda_1 K_1 + \cdots + \lambda_s K_s$ can be computed in polynomial time.*

The following Löwner–John-type "rounding lemmas" in terms of $\mathbb{B}^n$ and $C_n = [-1, 1]^n$ are due to [GLS88] and [AK90], respectively; see the survey [GK94, section 6.2] for some additional results in this context.

PROPOSITION 4. *There are oracle polynomial-time algorithms which accept as input a well-presented convex body K and construct affine transformations ϕ_1 and ϕ_2 such that $0 \in \operatorname{aff}(\phi_1(K))$, $0 \in \operatorname{aff}(\phi_2(K))$ and*

$$\operatorname{aff}(\phi_1(K)) \cap \mathbb{B}^n \subset \phi_1(K) \subset n\sqrt{n+1}\mathbb{B}^n, \qquad \operatorname{aff}(\phi_2(K)) \cap C_n \subset \phi_2(K) \subset 2(n+1)C_n.$$

1.3. Some estimates for numerical differentiation. As already mentioned above, computing (some/all) mixed volumes from the ordinary volume can be regarded as computing (some/all) of the coefficients of a polynomial from its values. This can in principle be done by numerical differentiation, and we will derive a few estimates now that will be used in section 3.

Let

$$q(x) = \sum_{i=0}^{n} c_i x^i$$

be a univariate polynomial of degree n, and let $\xi_0, \dots, \xi_n$ be pairwise different interpolation points. The *Lagrange-interpolation polynomials* $l_k(x) = \sum_{i=0}^{n} b_{ki} x^i$ on the

node set $X = \{\xi_0, \ldots, \xi_n\}$ satisfy

$$l_k(\xi_j) = \sum_{i=0}^{n} b_{ki}\xi_j^i = \delta_{jk},$$

where δ_{jk} is the usual Kronecker symbol. Then

$$q(x) = \sum_{k=0}^{n} q(\xi_k)l_k(x) = \sum_{i=0}^{n} \left(\sum_{k=0}^{n} q(\xi_k)b_{ki} \right) x^i;$$

hence

$$c_i = \sum_{k=0}^{n} b_{ki}q(\xi_k) \quad \text{for } i = 0, \ldots, n.$$

In general, only estimates $\hat{q}(\xi_j)$ of the function values $q(\xi_j)$ are available. In fact, for the purpose of section 3 we can only use estimates with bounded *relative* error. Here we suppose first that the *absolute* error is bounded beforehand, i.e., there is a positive δ such that

$$|\hat{q}(\xi_j) - q(\xi_j)| \leq \delta \quad \text{for } j = 0, \ldots, n.$$

Now, let $m \in \{0, \ldots, n\}$, and let us take

$$\hat{c}_m = \sum_{k=0}^{n} b_{km}\hat{q}(\xi_k)$$

as an estimate for c_m. Then we obtain

$$|c_m - \hat{c}_m| = \left| \sum_{k=0}^{n} b_{km}(q(\xi_k) - \hat{q}(\xi_k)) \right| \leq \delta \sum_{k=0}^{n} |b_{km}|. \tag{1.5}$$

Efficient methods for performing the computations in a systematical way (e.g., by using divided differences) can be found in any textbook on numerical analysis; see for example [BZ65]. The problem of how to choose the interpolation points to minimize the error terms $\sum_{k=0}^{n} |b_{km}|$ is discussed in (among others) [Ri75]; see also [MM85], [Sa74], [Ri90]; equidistant nodes are in general not optimal. We will use equidistant interpolation points anyway since, on the one hand, the subsequent analysis becomes more tractable and, on the other hand, the additional error introduced that way is dominated by other occurring error terms and hence is essentially irrelevant.

For the estimates in this subsection, we will normalize the nodes to the set $X = \{0, 1, \ldots, n\}$; in section 3 we will use the node set hX for some suitable positive rational h.

Let M denote the (infinite) Vandermonde matrix $M = (j^i)_{i,j\in\mathbb{N}_0}$, (with the setting $0^0 = 1$); for $r \in \mathbb{N}$ let $M^{(r)} = (j^i)_{i,j=0,\ldots,r-1}$ be the restriction of M to its first r rows and columns, and let $B^{(r)} = (b_{ij}^{(r)})_{i,j=0,\ldots,r-1}$ be the inverse of $M^{(r)}$. We will now derive an upper estimate for $\sum_{i=0}^{r-1} |b_{im}^{(r)}|$.

For $i, j \in \mathbb{N}$ with $i \geq j$, let σ_{ij} denote the *Stirling numbers of the second kind*, i.e., the number of partitions of the set $\{1, \ldots, i\}$ into j pairwise disjoint nonempty subsets (see, e.g., [St86]). In addition, let $\sigma_{00} = 1$, $\sigma_{i0} = 0$ for all $i > 0$ and $\sigma_{ij} = 0$

whenever $i < j$. Note that $j!\sigma_{ij}$ is the number of surjective mappings of $\{1,\dots,i\}$ into $\{1,\dots,j\}$; hence, it follows that

$$j^i = \sum_{k=0}^{\infty} \sigma_{ik} k! \left(\begin{array}{c} j \\ k \end{array} \right) = \sum_{k=0}^{\infty} \sigma_{ik} j(j-1)\cdot\ldots\cdot(j-k+1), \tag{1.6}$$

and, in particular,

$$\sigma_{ij} \le \frac{j^i}{j!}.$$

Thus, with the notation $(x)_k = x(x-1)(x-2)\cdot\ldots\cdot(x-k+1)$, the identity

$$x^i = \sum_{k=0}^{\infty} \sigma_{ik} (x)_k$$

holds for all integers $x = 0,\dots,i$ and hence holds for all $x \in \mathbb{R}$. Let

$$L = (\sigma_{ij})_{i,j\in\mathbb{N}_0}, \quad U = \left(\begin{array}{c} j \\ i \end{array} \right)_{i,j\in\mathbb{N}_0}, \quad \text{and} \quad D = \operatorname{diag}(0!, 1!, 2!, \dots).$$

Then L and U are an (infinite) lower and upper triangular matrix, respectively, and (1.6) can be written as

$$M = LDU.$$

Left multiplication by $L^{-1} = (s_{ij})_{i,j\in\mathbb{N}_0}$ yields

$$x(x-1)\cdot\ldots\cdot(x-i+1) = \sum_{k=0}^{\infty} s_{ik} x^k.$$

Hence, s_{ij} has sign $(-1)^{i-j}$ (for $j \le i$) and, evaluating the above identity for $x = -1$ yields

$$\sum_{j=0}^{i} |s_{ij}| = i!.$$

The numbers s_{ij} are called the *Stirling numbers of the first kind.* From

$$x^j = \sum_{i=0}^{\infty} \left(\begin{array}{c} j \\ i \end{array} \right) (x-1)^i \quad \text{and} \quad (x-1)^j = \sum_{i=0}^{\infty} \left(\begin{array}{c} j \\ i \end{array} \right) (-1)^{j-i} x^i,$$

we conclude for the inverse $U^{-1} = (w_{ij})_{i,j\in\mathbb{N}_0}$ of U that

$$w_{ij} = (-1)^{j-i} \binom{j}{i}.$$

Now, note that

$$\big(L^{(r)}\big)^{-1} = \big(L^{-1}\big)^{(r)}, \quad \big(D^{(r)}\big)^{-1} = \big(D^{-1}\big)^{(r)}, \quad \big(U^{(r)}\big)^{-1} = \big(U^{-1}\big)^{(r)}$$
$$\text{and} \quad M^{(r)} = L^{(r)} D^{(r)} U^{(r)}.$$

Hence,

$$B^{(r)} = \big(U^{(r)}\big)^{-1}\big(D^{(r)}\big)^{-1}\big(L^{(r)}\big)^{-1} = \big(U^{-1}\big)^{(r)}\big(D^{-1}\big)^{(r)}\big(L^{-1}\big)^{(r)},$$

and this reads explicitly as

(1.7)
$$b_{ij}^{(r)} = \sum_{k=0}^{r-1}(-1)^{k-i}\binom{k}{i}\frac{1}{k!}s_{kj} = (-1)^{i+j}\sum_{k=0}^{r-1}\binom{k}{i}\frac{1}{k!}|s_{kj}| \qquad (i,j=0,\dots,r-1).$$

This implies that for any $m \in \{0,\dots,r-1\}$,

$$\sum_{i=0}^{r-1}|b_{im}^{(r)}| = \sum_{i=0}^{r-1}\sum_{k=0}^{r-1}\binom{k}{i}\frac{|s_{km}|}{k!} \le \sum_{k=0}^{r-1}2^k\frac{|s_{km}|}{k!} < 2^r. \tag{1.8}$$

We conclude the univariate case with an additional technical estimate that is needed in section 3. It gives an upper bound on the error induced by using only $B^{(r)}$ (for some $r \le n$) rather than the full matrix $B^{(n+1)}$ in the computation of the coefficients of a polynomial of degree n.

Let, for $i,j \in \mathbb{N}_0$ with $j < r$,

$$d_{ij} = \sum_{k=0}^{r-1} b_{kj}^{(r)}k^i.$$

Clearly $d_{ij} = \delta_{ij}$ for $i < r$. For $i \ge r$, combining (1.6) and (1.7) yields

$$\begin{aligned}\left|\sum_{k=0}^{r-1} b_{kj}^{(r)}k^i\right| &= \left|\sum_{k=0}^{r-1}\left(\sum_{p=0}^{r-1}(-1)^{p-k}\binom{p}{k}\frac{1}{p!}s_{pj}\right)\left(\sum_{q=0}^{\infty}\sigma_{iq}q!\binom{k}{q}\right)\right| \\ &= \left|\sum_{p=0}^{r-1}\sum_{q=0}^{r-1}s_{pj}\sigma_{iq}\frac{q!}{p!}(-1)^{q-p}\left(\sum_{k=0}^{r-1}(-1)^{k-q}\binom{k}{q}\binom{p}{k}\right)\right| \\ &\le \sum_{p=0}^{r-1}|s_{pj}|\sigma_{ip} \le \sum_{p=0}^{r-1}p^i \le r^i.\end{aligned} \tag{1.9}$$

Let us close this section with a few brief remarks about the general multivariate case. Let, for $n,s \in \mathbb{N}$,

$$Y_{n,s} = \big\{y = (m_1,\dots,m_s) \in (\mathbb{N}_0)^s : \sum_{i=1}^{s} m_i = n\big\}.$$

Clearly,

$$N = |Y_{n,s}| = \binom{n+s-1}{n}.$$

Suppose that the elements of $Y_{n,s}$ are ordered (for instance lexicographically) so that $Y_{n,s} = \{y_1,\dots,y_N\}$, where $y_j = (m_{j,1},\dots,m_{j,s})$. Now we want to determine the coefficients of a homogeneous multivariate polynomial

$$q(x_1,\dots,x_s) = \sum_{j=1}^{N} c_j x_1^{m_{j,1}} \cdot \ldots \cdot x_s^{m_{j,s}}$$

from its function values. So, we have to choose N interpolation points in such a way that the $N \times N$ matrix

$$\left((\xi_1^{(i)})^{m_{j,1}} \cdot (\xi_2^{(i)})^{m_{j,2}} \cdot \ldots \cdot (\xi_s^{(i)})^{m_{j,s}} \right)_{i,j=1,\ldots,N},$$

a higher-dimensional analogue of the classical Vandermonde matrix, is nonsingular. (Note that, as opposed to the univariate case, it does not suffice to choose the N points mutually different.) Sufficient conditions for nonsingularity can be found in [CY77]; see also [Ol86]. In particular, one may take an $(s-1)$-dimensional simplex $S = \operatorname{conv}\{z_1, \ldots, z_s\}$ in $\mathbb{R}^s$ and choose $\xi^{(k_1,\ldots,k_s)} = \frac{1}{n} \sum_{j=1}^{s} k_j z_j$, where $(k_1, \ldots, k_s) \in Y_{n,s}$.

This implies, in particular, that when the dimension n is *fixed*, there is a polynomial-time algorithm which, given $s \in \mathbb{N}$ and ($\mathcal{V}$- or $\mathcal{H}$-) polytopes $P_1, \ldots, P_s$, computes all mixed volumes $V(P_{i_1}, \ldots, P_{i_n})$.

2. Deterministic algorithms. The present section discusses the problem of computing or approximating (mixed) volumes by means of deterministic algorithms. In particular we give results that focus on the difference of volume versus mixed volume computation.

2.1. Computing the volume of zonotopes. In this subsection we deal with the following problem.

VOLUME-OF-ZONOTOPES.

Given an $\mathcal{S}$-zonotope $Z = (n, s; c; z_1, \ldots, z_s)$, compute its volume.

Note that the problem asks for $\operatorname{vol}_n(Z)$, where $Z = c + \sum_{i=1}^{s} [0,1] z_i$. Since the volume is translation invariant, we can always assume that $c = 0$. Now, let A denote the $n \times s$ matrix with columns $z_1, \ldots, z_s$ and let $\mathcal{J}$ denote the family of all subsets I of $\{1, \ldots, s\}$ of cardinality n. Then (1.3) can be written in the form

$$\operatorname{vol}_n(Z) = \sum_{I \in \mathcal{J}} |\det B_I|,$$

where B_I is the $n \times n$-minor of A whose columns correspond to I. It is clear that for constant n or constant $s-n$, the number $\binom{s}{n}$ of $n \times n$ subdeterminants is polynomially bounded, whence the volume of zonotopes can be computed in polynomial time. The general case is, however, #$\mathbb{P}$-hard.

THEOREM 1. *VOLUME-OF-ZONOTOPES is #$\mathbb{P}$-hard.*

Proof. The proof will use a reduction of the following #$\mathbb{P}$-complete problem.

#SUBSET-SUM (see [GJ79], [Jo90]). *Given positive integers m, $\alpha_1, \ldots, \alpha_m$, and α, determine the number of different subsets J of $\{1, \ldots, m\}$ such that $\sum_{j \in J} \alpha_j = \alpha$.*

So, suppose $(m; \alpha_1, \ldots, \alpha_m, \alpha)$ is an instance of #SUBSET-SUM, and let $n = m+2$ and $s = 2m + 3$. Further, define

$$\begin{aligned}
z_{2k-1} &= e_k + \alpha_k e_{m+2}, && k = 1, \ldots, m;\\
z_{2k} &= e_k, && k = 1, \ldots, m+1;\\
z_{2m+1} &= e_{m+1} - \alpha e_{m+2};\\
z_{2m+3}^{\delta} &= -\sum_{i=1}^{m+1} e_i + \delta e_{m+2}, && \delta \in \{-1, 0, 1\},
\end{aligned}$$

where $e_1, \ldots, e_n$ denote again the standard basis vectors of $\mathbb{R}^n$, and set

$$Z_\delta = \sum_{i=1}^{s-1} [0,1] z_i + [0,1] z_s^{\delta}.$$

Suppose now that there is a polynomial-time algorithm $\mathcal{A}$ for solving the problem VOLUME-OF-ZONOTOPES, and apply $\mathcal{A}$ to compute $\mathrm{vol}_n(Z_{-1}) - 2\mathrm{vol}_n(Z_0) + \mathrm{vol}_n(Z_1)$. In terms of the determinant formula, this means that we are only interested in those $n \times n$ submatrices B_I of the $n \times s$ matrix

$$A_\delta = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 1 & \dots & 0 & 0 & 0 & 0 & -1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 1 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & 1 & -1 \\ \alpha_1 & 0 & \alpha_2 & 0 & \alpha_3 & 0 & \dots & \alpha_m & 0 & -\alpha & 0 & \delta \end{pmatrix}$$

which depend on δ. Then, clearly, B_I has to contain the last column z^δ_{2m+3} of A_δ, and in choosing the remaining $m+1$ columns, we have to select exactly one vector from each pair z_{2k-1}, z_{2k} $(k = 1, \dots, n-1)$. Therefore, the summands $|\det B_I|$ of the determinantal expansion of $\mathrm{vol}_n(Z_\delta)$ which are depending on δ are in one-to-one correspondence with the subsets J of $\{1, \dots, m+1\}$ via

$$j \in J \Longleftrightarrow 2j - 1 \in I.$$

From this it follows easily that there is an integer κ that depends only on $\alpha_1, \dots, \alpha_m$ and α but not on δ such that

$$\mathrm{vol}_n(Z_\delta) = \kappa + \sum_{J \subset \{1,\dots,m+1\}} |\delta + \sum_{i \in J} \alpha_j|,$$

where, for notational consistency, $\alpha_{m+1} = -\alpha$. Then,

$$\mathrm{vol}_n(Z_{-1}) - 2\mathrm{vol}_n(Z_0) + \mathrm{vol}_n(Z_1) = \sum_{J \subset \{1,\dots,m+1\}} \left(\Big|-1 + \sum_{j \in J} \alpha_j\Big| - 2\Big|\sum_{j \in J} \alpha_j\Big| + \Big|1 + \sum_{j \in J} \alpha_i\Big| \right).$$

Since for any nonzero integer γ,

$$|-1+\gamma| - 2|\gamma| + |1+\gamma| = 0,$$

it follows that

$$\frac{1}{2}\Big(\mathrm{vol}_n(Z_{-1}) - 2\mathrm{vol}_n(Z_0) + \mathrm{vol}_n(Z_1)\Big) = \Big|\big\{J \subset \{1, \dots, m+1\} : \sum_{j \in J} \alpha_i = 0\big\}\Big|.$$

But

$$\sum_{j \in J} \alpha_i = 0 \quad \text{if and only if} \quad m+1 \in J \quad \text{and} \quad \sum_{j \in J \cap \{1,\dots,m\}} \alpha_j = \alpha,$$

whence $\mathcal{A}$ gives rise to a polynomial-time algorithm for #SUBSET-SUM. □

Theorem 1 proves the #$\mathbb{P}$-hardness of evaluating $\sum_{I \in \mathcal{J}} |\det B_I|$. This result is in striking contrast to the fact that by the *Binet–Cauchy formula* (see, e.g., [BS83]),

$$\sum_{I \in \mathcal{J}} (\det B_I)^2 = \det(AA^T), \tag{2.1}$$

whence the sum of the *squares* of all $n \times n$ subdeterminants can be evaluated in polynomial time.

Note, further, that Proposition 1 can be applied to $\mathcal{S}$-zonotopes since it is standard fare to derive a well presentation for an $\mathcal{S}$-zonotope. So there is a polynomial-time randomized algorithm for VOLUME-OF-ZONOTOPES. Zonotopes come, however, with an additional structure (and in particular, with a natural dissection into parallelotopes) so it is conceivable that there are faster randomized algorithms for zonotopes than there are for general well-presented convex bodies. This question is, however, open.

For an easiness result complementing Theorem 1 see Theorem 6, and for an application of VOLUME-OF-ZONOTOPES to a problem in the oil industry see subsection 4.3.

We will now draw the first of a few consequences of Theorem 1 and prove a result that is relevant in subsection 2.4.

THEOREM 2. *The following problem VOLUME-OF-SUM-OF-ELLIPSOIDS is $\#\mathbb{P}$-hard: given $s, n \in \mathbb{N}$, nonsingular rational $(n \times n)$-matrices $A_1, \ldots, A_s$, an error bound $\epsilon \in \mathbb{Q}$, $\epsilon > 0$, compute a rational number $\hat{V}$ which satisfies*

$$\left| \hat{V} - \mathrm{vol}_n(E_1 + E_2 + \cdots + E_s) \right| < \epsilon,$$

where E_i is the ellipsoid $E_i = \{x \in \mathbb{R}^n : x^T A_i^T A_i x \leq 1\}$.

Proof. Let $(n, s; c; z_1, \ldots, z_s)$ be an instance of VOLUME-OF-ZONOTOPES and set $Z = \sum_{i=1}^{s} [-1, 1] z_i$. Note that

$$\mathrm{vol}_n(Z) = 2^n \mathrm{vol}_n \left(c + \sum_{i=1}^{s} [0, 1] z_i \right),$$

whence it suffices to show how the computation of $\mathrm{vol}_n(Z)$ can be reduced to a suitable instances of VOLUME-OF-SUM-OF-ELLIPSOIDS.

For each $i = 1, \ldots, s$ we compute first an orthogonal basis $\{v_{i,1}, \ldots, v_{i,n}\}$ of $\mathbb{R}^n$ such that $v_{i,1} = z_i$. Let B_i be the $n \times n$-matrix with rows $v_{i,1}^T, \ldots, v_{i,n}^T$, set for $\mu \in \mathbb{N}$,

$$D_i^\mu = \mathrm{diag}\left(\frac{1}{\langle z_i, z_i \rangle}, \frac{\mu}{\langle v_{i,2}, v_{i,2} \rangle}, \ldots, \frac{\mu}{\langle v_{i,n}, v_{i,n} \rangle} \right),$$

and define the ellipsoid

$$E_i^\mu = \{x \in \mathbb{R}^n : x^T (D_i^\mu B_i)^T (D_i^\mu B_i) x \leq 1\}.$$

Then we have

$$[-1, 1] z_i \subset E_i^\mu \subset [-1, 1] z_i + \frac{1}{\mu} \sum_{j=2}^{n} [-1, 1] v_{i,j}.$$

Now, let $Z' = \sum_{i=1}^{s} \sum_{j=2}^{n} [-1, 1] v_{i,j}$, and let $R \in \mathbb{N}$ such that $Z \cup Z' \subset RC_n$, where C_n denotes again the standard unit cube. Then the above inclusions yield

$$Z \subset E_1^\mu + E_2^\mu + \cdots + E_s^\mu \subset Z + \frac{1}{\mu} Z' \subset Z + \frac{R}{\mu} C_n.$$

Now note that for any $\lambda > 0$,

$$\mathrm{vol}_n(Z + \lambda C_n) - \mathrm{vol}_n(Z) = \sum_{i=1}^{n} \binom{n}{i} V(\overbrace{Z, \ldots, Z}^{n-i}, \overbrace{C_n, \ldots, C_n}^{i}) \lambda^i,$$

and this implies that

$$\mathrm{vol}_n(E_1^\mu + \cdots + E_s^\mu) - \mathrm{vol}_n(Z) \le \sum_{i=1}^{n} \binom{n}{i} V(\overbrace{Z,\ldots,Z}^{n-i}, \overbrace{C_n,\ldots,C_n}^{i}) \left(\frac{R}{\mu}\right)^i \le \frac{(4R)^n}{\mu}.$$

Hence, if for $\mu_0 = \lceil \frac{2}{\epsilon}(4R)^n \rceil$ the volume of $E_1^{\mu_0} + \cdots + E_s^{\mu_0}$ is approximated to absolute error $\frac{\epsilon}{2}$, we obtain an estimate of $\mathrm{vol}_n(Z)$ to absolute error ϵ. Further, it follows from (1.3) that $size(\mathrm{vol}_n(Z))$ is bounded by a polynomial in the input size. Therefore, it suffices to approximate $\mathrm{vol}_n(Z)$ to a sufficiently small absolute error ϵ whose size is polynomially bounded and then perform the usual rounding (with continued fractions) in order to obtain $\mathrm{vol}_n(Z)$ precisely.

Finally note that all constructions and computations can be done in polynomial time; this completes the transformation. □

2.2. Mixed volumes of paralellotopes. We give some hardness results for computing mixed volumes of parallelotopes. The first involves axes-parallel parallelotopes which (for brevity) will be called *boxes*.

Before we state the result we need two lemmas.

LEMMA 1. *Let the entries of* $A = (\alpha_{ij})_{i,j=1,2,,\ldots,n}$ *be nonnegative rationals, and for* $i = 1,\ldots,n$ *set* $Z_i = \sum_{j=1}^{n} [0, \alpha_{ij}] e_j$. *Then*

$$n! V(Z_1,\ldots,Z_n) = \mathrm{per}(A),$$

where $\mathrm{per}(A)$ *denotes the permanent of* A.

Proof. Note that the Z_i are all boxes, and so is $\sum_{i=1}^{n} \lambda_i Z_i$ for each n-tuple $(\lambda_1,\ldots,\lambda_n)$ of nonnegative reals. Hence,

$$\mathrm{vol}_n\left(\sum_{i=1}^{n} \lambda_i Z_i\right) = \mathrm{vol}_n\left(\sum_{j=1}^{n} \left[0, \sum_{i=1}^{n} \lambda_i \alpha_{ij}\right] e_j\right) = \prod_{j=1}^{n} \left(\sum_{i=1}^{n} \lambda_i \alpha_{ij}\right).$$

Comparing the coefficients of $\lambda_1 \cdot \lambda_2 \cdot \ldots \cdot \lambda_n$ we see that

$$V(Z_1,\ldots,Z_n) = \frac{1}{n!} \sum_{j_1=1}^{n} \cdots \sum_{j_n=1}^{n} \epsilon_{j_1,\ldots,j_n} \alpha_{1,j_1} \cdot \ldots \cdot \alpha_{n,j_n},$$

where

$$\epsilon_{j_1,\ldots,j_n} = \begin{cases} 1 & \text{if } j_1,\ldots,j_n \text{ is a permutation of } 1,2,\ldots,n, \\ 0 & \text{otherwise,} \end{cases}$$

and this proves the assertion. □

The problem of computing the permanent of 0-1-matrices is known to be #ℙ-complete [Va77]; see also [Va79], [Br86], [JS89], [LS90], [KKLLL93]. Hence, Lemma 1 implies that the problem of computing the mixed volume $V(Z_1,\ldots,Z_n)$ of n faces Z_i of the cube $[0,1]^n$ is also #ℙ-complete. (Note, on the positive side, that in view of Lemma 1, Theorem 11 yields a randomized polynomial-time algorithm for estimating the permanent of certain classes of matrices.) In order to extend this result to *proper* boxes, observe that if we replace the 0-entries of a given 0-1-matrix B by a parameter α to obtain an α-1-matrix B_α, then $\mathrm{per}(B_\alpha)$ is a polynomial in α; evaluation of this polynomial for $n+1$ different values of α (or for one sufficiently small value

of α) allows us to compute its constant term $\mathrm{per}(B)$. In order to prove the sharper statement of Theorem 3, we use the following strengthening of [Va77]'s hardness result to the permanent of α-β-matrices with prescribed α and β.

LEMMA 2. *The following problem is $\#\mathbb{P}$-hard for any pair α, β of (fixed) distinct rationals: given a positive integer n, and an $n \times n$ matrix A with entries α, β, compute* $\mathrm{per}(A)$.

Proof. We may assume that $\beta = \alpha + 1$ and $\alpha \neq 0$. Let $B = (b_{ik})_{i,k=1,\ldots,n}$ be an arbitrary 0-1-matrix. We will reduce the computation of $\mathrm{per}(B)$ to the computation of the permanent of several matrices with $\alpha, \alpha + 1$ entries.

Let G denote the bipartite graph on $2n$ vertices whose adjacency matrix is B, and for $k = 1, \ldots, n$ let M_k be the number of matchings of size k in G. We want to compute $M_n = \mathrm{per}(B)$. For $j = 0, \ldots, n$ let $X^{(j)} = (x_{ik}^{(j)})$, denote the $(n+j) \times (n+j)$ matrix with entries

$$x_{ik}^{(j)} = \begin{cases} \alpha + b_{ik} & \text{for } i, k = 1, \ldots, n; \\ \alpha & \text{otherwise.} \end{cases}$$

Clearly, $X^{(j)}$ has only entries $\alpha, \alpha + 1$; whence using an oracle for evaluating the permanent of matrices with $\alpha, \alpha+1$ entries $n+1$ times, we can determine the permanents of all the $X^{(j)}$. On the other hand, we have

$$\sum_{k=0}^{n} M_k (n - k + j)! \alpha^{n-k+j} = \mathrm{per}(X^{(j)}) \qquad (j = 0, \ldots, n). \tag{2.2}$$

To see this, regard α as an indeterminate, and expand $\mathrm{per}(X^{(j)})$ as a polynomial in α. Then the terms contributing to the coefficient of α^{n-k+j} arise as follows. For every k-matching in G we obtain a product $(\alpha + 1)^k$, and this can be completed in $(n - k + j)!$ ways to give a lowest term α^{n-k+j}. We do this by selecting the α term from the product of monomials (either α or $\alpha + 1$) represented by any matching on the complete bipartite graph induced by the remaining $n - k + j$ rows and columns.

Therefore, if the above system (2.2) of linear equations is nonsingular, we can solve it for M_n, and this establishes the $\#\mathbb{P}$-hardness result.

To see that (2.2) is indeed nonsingular, let us rewrite the system as follows:

$$\sum_{k=0}^{n} \binom{k+j}{j} (k! \alpha^k M_{n-k}) = \frac{j! \, \mathrm{per}(X^{(j)})}{\alpha^j} \qquad (j = 0, \ldots, n).$$

Introducing the new variables $y_k = k! \alpha^k M_{n-k}$, the question now reduces to deciding whether the $(n + 1) \times (n + 1)$ matrix C with entries $c_{kj} = \binom{k+j}{j}$ is nonsingular. But this follows easily from the Vandermonde identity

$$\sum_{r=0}^{n} \binom{k}{r} \binom{j}{r} = \binom{k+j}{j} \quad \text{for } k, j \leq n,$$

since $C = UU^T$, where U is the lower triangular matrix with entries $u_{ij} = \binom{i}{j}$ which has all its diagonal elements 1. ∎

Now we can prove Theorem 3.

THEOREM 3. *Let α, β be (fixed) distinct positive rationals. Then the following restriction of MIXED-VOLUME-OF-BOXES is $\#\mathbb{P}$-hard: given $n \in \mathbb{N}$, and for $i, j =$*

$1, 2, , \ldots, n$, *an element* $\alpha_{i,j}$ *of* $\{\alpha, \beta\}$, *compute the mixed volume* $V(Z_1, \ldots, Z_n)$ *for the proper boxes* $Z_i = \sum_{j=1}^{n} [0, \alpha_{i,j}] e_j$, $(i = 1, \ldots, n)$.

Proof. The result is a simple consequence of Lemmas 1 and 2. □

Theorem 3 implies directly the "instability" result that, while the case of $\epsilon = 0$ is trivial, it is #ℙ-hard for any $\epsilon > 0$ to compute the mixed volume of n proper boxes, all containing the unit cube C_n and all being contained in the cube $(1 + \epsilon)C_n$.

Note that MIXED-VOLUME-OF-BOXES can be solved in polynomial time if the number s of different boxes is bounded beforehand. In this case there are only $O(n^{s-1})$ different mixed volumes, and they can all be computed by the approach of subsection 1.3 (see [GK94, subsection 4.1]), since their Minkowski sum is a box whose volume can be computed easily. We will show, however, that the corresponding problem for just two proper rectangular parallelotopes is #ℙ-hard if they are not both required to be axes-parallel.

THEOREM 4. *The following problem is* #ℙ*-hard: given* $n \in \mathbb{N}$, $m \in \{0, \ldots, n\}$, $\alpha_1, \ldots, \alpha_n \in \mathbb{N}$, *and integer vectors* $y_1, \ldots, y_n$ *which form an orthogonal basis of* $\mathbb{R}^n$, *compute* $V(\overbrace{Z_1, \ldots, Z_1}^{n-m}, \overbrace{Z_2, \ldots, Z_2}^{m})$, *where* $Z_1 = \sum_{i=1}^{n} [0, 1] y_i$ *and* $Z_2 = \sum_{i=1}^{n} [0, \alpha_i] e_i$.

Proof. We use the problem VOLUME-OF-ZONOTOPES of Theorem 1 for a reduction. Let $Z = (n, s; c, z_1, \ldots, z_s)$ be an $\mathcal{S}$-zonotope. We may assume without loss of generality that $z_1, \ldots, z_s \in \mathbb{Z}^n$, that $s > n$, and that Z is proper. Now, let A denote the $n \times s$ matrix with columns $z_1, \ldots, z_s$. Since, by (1.3), elementary row operations to A do not change the volume of the zonotope generated by the columns of A, we may further assume that the rows $v_1, \ldots, v_n$ of A are orthogonal.

Let $\{v_{n+1}, \ldots, v_s\} \subset \mathbb{Q}^s$ be an orthogonal basis of the orthogonal complement of the linear hull of $\{v_1, \ldots, v_n\}$ such that the sizes of $v_{n+1}, \ldots, v_s$ are bounded by a polynomial in $size(Z)$. Note that such a basis can be computed essentially by solving a system of linear equations. Let B denote the $s \times s$ matrix that is obtained from A by augmenting the rows $v_{n+1}, \ldots, v_s$, and let $y_1, \ldots, y_s$ be the column vectors of B. Since the rows of A are orthogonal, so are the columns. Hence,

$$Z_1 = \sum_{i=1}^{s} [0, 1] y_i$$

is a proper rectangular parallelotope in $\mathbb{R}^s$. Set, further,

$$C = \sum_{i=n+1}^{s} [0, 1] e_i, \qquad \text{and for } 0 < \mu < 1, \quad Z_2^{\mu} = C + \mu \sum_{i=1}^{n} [0, 1] e_i.$$

By Proposition 2, applied with $U = \{0\}^n \times \mathbb{R}^{s-n}$ (and hence $U^{\perp} = \mathbb{R}^n \times \{0\}^{s-n}$), we obtain

$$\binom{s}{n} V(\overbrace{Z_1, \ldots, Z_1}^{n}, \overbrace{C, \ldots, C}^{s-n}) = \mathrm{vol}_n(Z),$$

and this gives already an #ℙ-hardness result for the case that one of the parallelotopes is permitted to be lower dimensional. To complete the proof of Theorem 4, observe that (by (1.4))

$$V(\overbrace{Z_1, \ldots, Z_1}^{n}, \overbrace{Z_2^{\mu}, \ldots, Z_2^{\mu}}^{s-n}) = \sum_{i=0}^{s-n} \binom{s-n}{i} V(\overbrace{Z_1, \ldots, Z_1}^{n}, \overbrace{C, \ldots, C}^{s-n-i}, \overbrace{\hat{C}, \ldots, \hat{C}}^{i}) \mu^i,$$

where $\hat{C} = [0,1]^n \times \{0\}^{s-n}$. Since there is a positive integer R of size bounded by a polynomial in $size(Z)$ such that $Z_1 \subset R[0,1]^s$, it follows that

$$V(\overbrace{Z_1,\dots,Z_1}^{n},\overbrace{C,\dots,C}^{s-n}) \le V(\overbrace{Z_1,\dots,Z_1}^{n},\overbrace{Z_2^{\mu},\dots,Z_2^{\mu}}^{s-n}) \le V(\overbrace{Z_1,\dots,Z_1}^{n},\overbrace{C,\dots,C}^{s-n}) + 2^{s-n}R^n\mu.$$

Now, let μ_0 be a positive rational of size bounded by a polynomial in the input size such that $1/\mu_0 > 2 \cdot 2^{s-n}R^n\binom{s}{n}$, and set $Z_2 = Z_2^{\mu_0}$. Then Z_2 is a proper rectangular parallelotope, and

$$\left|\mathrm{vol}_n(Z) - \binom{s}{n} V(\overbrace{Z_1,\dots,Z_1}^{n},\overbrace{Z_2,\dots,Z_2}^{s-n})\right| < \frac{1}{2}.$$

Since $\mathrm{vol}_n(Z)$ is an integer, this shows that it suffices to compute the mixed volume $V(\overbrace{Z_1,\dots,Z_1}^{n},\overbrace{Z_2,\dots,Z_2}^{n-s})$ in order to obtain $\mathrm{vol}_n(Z)$. To conclude the transformation just apply a suitable scaling to make μ_0 integer. □

As a simple consequence of Theorem 4 we can derive a sharpening of Theorem 1.

THEOREM 5. *The following problem is #ℙ-hard: given $n \in \mathbb{N}$, and two n-tuples $v_1,\dots,v_n$ and $w_1,\dots,w_n$ of integer vectors that form orthogonal bases of $\mathbb{R}^n$, compute the volume of the Minkowski sum*

$$\mathrm{vol}_n\left(\left(\sum_{i=1}^{n}[0,1]v_i\right) + \left(\sum_{j=1}^{n}[0,1]w_j\right)\right).$$

Proof. Let $Z_1 = \sum_{i=1}^{n}[0,1]v_i$ and $Z_2 = \sum_{j=1}^{n}[0,1]w_j$. For the proof of the theorem, just note that all mixed volumes of Z_1 and Z_2 can be computed by the method indicated in subsection 1.3 by evaluating $\mathrm{vol}_n(Z_1 + \xi Z_2)$ for $n+1$ mutually disjoint interpolation points $\xi_0,\dots,\xi_n$, and apply Theorem 4. □

2.3. Easiness of mixed volume computation. The results of the previous subsection show that mixed volume computation is in general at least as hard as any problem in #ℙ. The present subsection addresses the question of whether computing mixed volumes is possibly even harder.

As shown in [DF88], using any oracle which solves some #ℙ-complete problem in constant time, the volume of a $\mathcal{V}$-polytope can be computed in polynomial time; this is stated by saying that volume computation for $\mathcal{V}$-polytopes is #ℙ-*easy*.

$\mathcal{H}$-presented polytopes come with the additional difficulty that the size of their volume is not bounded by a polynomial in the input size. An example was given by [La91], showing that there is no polynomial-space algorithm for *exact* computation of the volume of $\mathcal{H}$-polytopes. However, approximation to any positive rational absolute error ϵ is again #ℙ-easy for $\mathcal{H}$-polytopes, [DF88].

It is clear from section 1.3 (see also [GK94]) that the easiness results for computing or approximating the volume can be extended to mixed volumes if the number s of sets under consideration is bounded beforehand. If, however, s is part of the input the number of volume computations needed for the numerical differentiation approach to compute a single mixed volume cannot be bounded by a polynomial in n and s. The

reason is that this method "essentially" computes all mixed volumes at once and their number is exponential.

We will show in the following, however, that even in this general case computation (for $\mathcal{V}$-polytopes or $\mathcal{S}$-zonotopes) or approximation (for $\mathcal{H}$-polytopes) of any single mixed volume is $\#\mathbb{P}$-easy. We begin with the easier case of $\mathcal{S}$-zonotopes.

THEOREM 6. *Let Π be any $\#\mathbb{P}$-complete problem. Then any oracle $\mathcal{O}_\Pi$ for solving Π can be used to produce an algorithm that runs in time that is oracle-polynomial in the input size for solving the following problem:*

Instance: *$n, s \in \mathbb{N}$, and $m_1, \ldots, m_s \in \mathbb{N}$ such that $\sum_{i=1}^s m_i = n$, $\mathcal{S}$-zonotopes $Z_i = (n, s_i; c_i; z_{i,1}, \ldots, z_{i,s_i})$, for $i = 1, \ldots, s$.*

Task: *Compute the mixed volume*

$$V(\overbrace{Z_1, \ldots, Z_1}^{m_1}, \overbrace{Z_2, \ldots, Z_2}^{m_2}, \ldots, \overbrace{Z_s, \ldots, Z_s}^{m_s}).$$

Proof. The proof reduces the problem to the task of approximating the volume of a (typically nonconvex) finite union of parallelotopes.

For $i \in S = \{1, \ldots, s\}$, let $J_i = \{(i,1), \ldots, (i, s_i)\}$, set $J = J_1 \cup \cdots \cup J_s$, and

$$\mathcal{J}_{m_1,\ldots,m_s} = \{I \subset J : |I \cap J_i| = m_i, \text{ for } i \in S\}.$$

Further, let $r = \sum_{i=1}^s s_i$, and let A denote the $n \times r$ matrix

$$A = (z_{1,1}, \ldots, z_{1,s_1}, \ldots, z_{s,1}, \ldots, z_{s,s_s}).$$

Then it is easy to see, by expanding $\mathrm{vol}_n(\sum_{i=1}^s \lambda_i Z_i)$, that

$$\binom{n}{m_1, \ldots, m_s} V(\overbrace{Z_1, \ldots, Z_1}^{m_1}, \ldots, \overbrace{Z_s, \ldots, Z_s}^{m_s}) = \sum_{I \in \mathcal{J}_{m_1,\ldots,m_s}} |\det B_I|, \tag{2.3}$$

where B_I denotes the $n \times n$ submatrix of A with column indices in I, and $\binom{n}{m_1,\ldots,m_s}$ is the usual multinomial coefficient, i.e.,

$$\binom{n}{m_1, \ldots, m_s} = \frac{n!}{m_1! \cdot \ldots \cdot m_s!}.$$

To prove the theorem, we will now interpret (2.3) geometrically. In fact, let

$$Z = \sum_{(i,j) \in J} [0,1] z_{i,j},$$

and let again $\mathcal{J}$ denote the family of all subsets I of J of cardinality n. Using a simple inductive argument (with respect to r), we see that there is a subset $\mathcal{I}$ of $\mathcal{J}$ and that there are vectors p_I $(I \in \mathcal{I})$ such that the parallelotopes

$$P_I = p_I + \sum_{i \in I} [0,1] z_i \qquad (I \in \mathcal{I})$$

form a dissection of Z into proper parallelotopes; see [Sh74]. Further, for each $x \in Z \cap \mathbb{Q}^n$, a subset $I \in \mathcal{I}$ with $x \in P_I$ can be found in time bounded by a polynomial in $size(Z)$ and $size(x)$. Note that these parallelotopes are in one-to-one correspondence with the nonsingular matrices B_I with $I \in \mathcal{J}$. Hence, with $Z_{m_1,\ldots,m_s} = \bigcup_{I \in \mathcal{J}_{m_1,\ldots,m_s}} P_I$, we have

$$\binom{n}{m_1, \ldots, m_s} V(\overbrace{Z_1, \ldots, Z_1}^{m_1}, \ldots, \overbrace{Z_s, \ldots, Z_s}^{m_s}) = \mathrm{vol}_n(Z_{m_1,\ldots,m_s}),$$

and membership in $Z_{m_1,\ldots,m_s}$ of a point $x \in \mathbb{Q}^n$ can be checked in polynomial time.

Now, let $R = \sum_{(i,j)\in J} \|z_{i,j}\|_\infty$, whence $Z \subset R[-1,1]^n$. Further, let ϵ be a positive rational, let

$$\alpha = \left\lceil \frac{2r^n n^2 (2R)^n}{\epsilon} \right\rceil, \quad \text{and} \quad \delta = \frac{R}{\alpha}.$$

For each integer vector $t = (\tau_1, \ldots, \tau_n)$, let

$$x_t = \delta \left(\tau_1 + \frac{1}{2}, \ldots, \tau_n + \frac{1}{2}\right)^T, \quad \text{and} \quad C_t = x_t + \frac{\delta}{2}[-1,1]^n.$$

For each x_t, membership in $Z_{m_1,\ldots,m_s}$ can be decided in polynomial time, so $\mathcal{O}_\Pi$ can be used to construct a counting machine that outputs the number N of integer vectors t for which $x_t \in Z_{m_1,\ldots,m_s}$. So, if ν is the number of cubes C_t that intersect the boundary of $Z_{m_1,\ldots,m_s}$, we have

$$|N\delta^n - \mathrm{vol}_n(Z_{m_1,\ldots,m_s})| \le \nu\delta^n.$$

It is readily seen that each facet of any Z_I ($I \in \mathcal{J}_{m_1,\ldots,m_s}$) is intersected by at most $2n(2\alpha)^{n-1}$ such cubes, whence (after some standard calculations)

$$|N\delta^n - \mathrm{vol}_n(Z_{(m_1,\ldots,m_s)})| \le 4r^n n^2 (2\alpha)^{n-1}\delta^n \le \epsilon \le \binom{n}{m_1,\ldots,m_s}\epsilon.$$

Therefore,

$$\left| V(\overbrace{Z_1,\ldots,Z_1}^{m_1},\ldots,\overbrace{Z_s,\ldots,Z_s}^{m_s}) - N\delta^n \binom{n}{m_1,\ldots,m_s}^{-1} \right| \le \epsilon.$$

Now, $size(\mathrm{vol}_n(Z_{m_1,\ldots,m_s}))$ is bounded above by a polynomial in the size of the input. So a suitable choice of ϵ and subsequent rounding yields $V(\overbrace{Z_1,\ldots,Z_1}^{m_1},\ldots,\overbrace{Z_s,\ldots,Z_s}^{m_s})$ exactly. □

Note that as a corollary we see that VOLUME-OF-ZONOTOPES is #ℙ-easy.

THEOREM 7. *Let Π be any #ℙ-complete problem. Then any oracle $\mathcal{O}_\Pi$ for solving Π can be used to produce an algorithm that runs in time that is oracle-polynomial in the input size (including $size(\epsilon)$ in the second case) for solving the following problems:*

Instance 1: *$n, s \in \mathbb{N}$ and $m_1, \ldots, m_s \in \mathbb{N}$ such that $\sum_{i=1}^s m_i = n$, $\mathcal{V}$-polytopes $P_i = (n, n_i; v_{i,1}, \ldots, v_{i,n_i})$, for $i = 1, \ldots, s$.*

Task 1: *Compute the mixed volume*

$$V(\overbrace{P_1,\ldots,P_1}^{m_1},\overbrace{P_2,\ldots,P_2}^{m_2},\ldots,\overbrace{P_s,\ldots,P_s}^{m_s}).$$

Instance 2: *$n, s \in \mathbb{N}$ and $m_1, \ldots, m_s \in \mathbb{N}$ such that $\sum_{i=1}^s m_i = n$, $\mathcal{H}$-polytopes $P_i = (n, n_i; A_i, b_i)$, for $i = 1, \ldots, s$, a positive rational number ϵ.*

Task 2: *Compute a rational number $\hat{V}_{m_1,m_2,\ldots,m_s}$ such that*

$$\left| \hat{V}_{m_1,m_2,\ldots,m_s} - V(\overbrace{P_1,\ldots,P_1}^{m_1},\overbrace{P_2,\ldots,P_2}^{m_2},\ldots,\overbrace{P_s,\ldots,P_s}^{m_s}) \right| \le \epsilon.$$

Proof. For $\mathcal{V}$- or $\mathcal{H}$-polytopes it is not so clear (as it was for $\mathcal{S}$-zonotopes) that mixed volumes can be reduced to a volume computation, yet it is possible. The proof makes substantial use of a formula of [Sc94] (a generalization of [Be92]), and we will begin by restating [Sc94]'s approach.

For a polytope P in some $\mathbb{R}^m$ and an integer k with $0 \le k \le m$, let, as usual, $\mathcal{F}_k(P)$ denote the set of k-faces of P, and let $\mathcal{F}(P) = \bigcup_{k=0}^{m} \mathcal{F}_k(P)$. Further, for a face F of P, let $N(P,F)$ denote the cone of outer normals of P at F.

Now, let $P_1, \dots, P_s$ be polytopes in $\mathbb{R}^n$. Set $r = s \cdot n$ and

$$\tilde{P} = P_1 \times P_2 \times \cdots \times P_s \subset \overbrace{\mathbb{R}^n \times \mathbb{R}^n \times \cdots \times \mathbb{R}^n}^{s} = \mathbb{R}^r.$$

It is easy to see that

$$\mathcal{F}(\tilde{P}) = \{F_1 \times F_2 \times \cdots \times F_s \ : F_1 \in \mathcal{F}(P_1), \dots, F_s \in \mathcal{F}(P_s)\}$$

and that

$$\mathcal{F}_k(\tilde{P}) = \bigcup_{\substack{k_1,\dots,k_s \in \mathbb{N}_0 \\ k_1+\cdots+k_s=k}} \{F_1 \times F_2 \times \cdots \times F_s : F_1 \in \mathcal{F}_{k_1}(P_1), \dots, F_s \in \mathcal{F}_{k_s}(P_s)\}.$$

Let

$$\Delta = \{(x^T, x^T, \dots, x^T)^T \in \mathbb{R}^r : x \in \mathbb{R}^n\};$$

Δ is a linear subspace of $\mathbb{R}^r$ of dimension n. For $\tilde{v} = (v_1, \dots, v_r)^T \in \Delta^\perp \setminus \{0\}$, let $\Delta_{\tilde{v}} = \mathrm{lin}(\Delta \cup \{\tilde{v}\})$, and let

$$\Delta_{\tilde{v}}^+ = \{\tilde{w} \in \Delta_{\tilde{v}} : \langle \tilde{w}, \tilde{v} \rangle > 0\},$$

the corresponding "positive" open halfspace. Further, let π_Δ and $\pi_{\Delta_{\tilde{v}}}$ denote the orthogonal projections onto Δ and $\Delta_{\tilde{v}}$, respectively, let π'_Δ be the restriction of π_Δ to the set $\Delta_{\tilde{v}}$, and set $P_{\tilde{v}} = \pi_{\Delta_{\tilde{v}}}(\tilde{P})$. Note that

$$\pi_\Delta(x_1, \dots, x_s) = \frac{1}{s}\left(\sum_{i=1}^{s} x_i^T, \dots, \sum_{i=1}^{s} x_i^T\right)^T.$$

Then $\mathrm{vol}_n(\pi_\Delta(\tilde{P}))$ is just the sum of the volumes of the projections of those facets of $P_{\tilde{v}}$ with outer normal vector $\tilde{w}$ in $\Delta_{\tilde{v}}^+$.

Suppose that none of the $\tilde{w} \in \Delta_{\tilde{v}}^+$ is orthogonal to a hyperplane in $\mathbb{R}^r$ that supports $\tilde{P}$ in a face of dimension greater than n. Then each facet of $P_{\tilde{v}}$ with outer normal $\tilde{w}$ in $\Delta_{\tilde{v}}^+$ is the projection of exactly one n-dimensional face $\tilde{F} \in \mathcal{F}_n(\tilde{P})$ such that $\tilde{w} \in N(\tilde{P}, \tilde{F})$. Let $\tilde{\mathcal{F}}_n^+$ be the set of all faces $\tilde{F} \in \mathcal{F}_n(\tilde{P})$ for which

$$N(\tilde{P}, \tilde{F}) \cap \Delta_{\tilde{v}}^+ \neq \emptyset.$$

It follows that

$$\pi_\Delta(\tilde{P}) = \bigcup_{\tilde{F} \in \tilde{\mathcal{F}}_n^+} \pi_\Delta(\tilde{F})$$

and

$$\mathrm{vol}_n(\pi_\Delta(\tilde{F}) \cap \pi_\Delta(\tilde{G})) = 0 \quad \text{for all } \tilde{F}, \tilde{G} \in \tilde{\mathcal{F}}_n^+,\ \tilde{F} \neq \tilde{G}.$$

Now, let

$$F_{m_1,\dots,m_s} = \bigcup_{\substack{\tilde{F} = F_1 \times F_2 \times \cdots \times F_s \in \tilde{\mathcal{F}}_n^+ \\ \dim(F_1)=m_1,\dots,\dim(F_s)=m_s}} \mathrm{int}(F_1 + \cdots + F_s),$$

where int is taken with respect to $\mathbb{R}^n$. (Clearly, in terms of volume computations, taking the interior does not matter, and we do it only for technical reasons that become clear when we develop a method for checking membership in $F_{m_1,\dots,m_s}$ later.) By replacing P_i by $\lambda_i P_i$ for $\lambda_i > 0$, and comparing coefficients we obtain [Sc94]'s formula

$$\binom{n}{m_1,\dots,m_s} V(\overbrace{P_1,\dots,P_1}^{m_1},\dots,\overbrace{P_s,\dots,P_s}^{m_s}) = \mathrm{vol}_n(F_{m_1,\dots,m_s}) = \sum_{\substack{\tilde{F}=F_1\times F_2\times\cdots\times F_s\in\tilde{\mathcal{F}}_n^+\\ \dim(F_1)=m_1,\dots,\dim(F_s)=m_s}} \mathrm{vol}_n(F_1+\cdots+F_s).$$

Suppose that rational vectors $v_1, v_2, \dots, v_s \in \mathbb{R}^n$ can be computed in polynomial time with $\tilde{v} = (v_1^T, v_2^T, \dots, v_s^T)^T \in \Delta^\perp$ and such that no $\tilde{w} \in \Delta_{\tilde{v}}^+$ supports $\tilde{P}$ in a face of dimension greater than n. We can then apply the same proof technique as in the proof of Theorem 6, if we can check membership of a point z in $F_{m_1,\dots,m_s}$ in polynomial time. But this can be done as follows (in both cases where $P_1, \dots, P_s$ are $\mathcal{V}$- or $\mathcal{H}$-polytopes).

Given $z \in \mathbb{R}^n$, we first check whether $z \in P_1 + P_2 + \cdots + P_s$. Clearly, this can be done by linear programming. If the answer is affirmative, we compute the vector $\tilde{z}_0$ that is given by

$$\{\tilde{z}_0\} = \{\tilde{z} + \lambda\tilde{v} : \lambda \geq 0\} \cap \mathrm{relbd}(P_{\tilde{v}}), \quad \text{where } \tilde{z} = (z^T, \dots, z^T)^T.$$

To see that this can be done in polynomial time, observe that the corresponding parameter λ_0 is the solution of the linear program

$$\max\langle \tilde{v}, \tilde{x}\rangle \quad \text{s.t.} \quad \tilde{x} \in \tilde{P} \cap (\tilde{z} + \Delta^\perp).$$

Since $\tilde{P} = \{\tilde{x} = (x_1^T, \dots, x_s^T)^T : x_1 \in P_1, \dots, x_s \in P_s\}$ and $\tilde{z} + \Delta^\perp$ can be easily expressed in the form $A\tilde{x} = b$, where A is an $n \times r$ matrix with 0-1 coefficients, $b \in \mathbb{Q}^n$, and the size is bounded by a polynomial in r and $size(z)$, the given linear program can be solved in polynomial time.

Now, if $\lambda_0 = 0$ we know that $z \in \mathrm{bd}(P_1 + P_2 + \cdots + P_s)$, and we report that $z \notin F_{m_1,\dots,m_s}$.

Otherwise we compute an outer normal $\tilde{w} \in \Delta_{\tilde{v}}^+$ of $P_{\tilde{v}}$ at $\tilde{z}_0$. This can be done in polynomial time.

Let $F_{\tilde{w}}$ denote the face of $P_{\tilde{v}}$ that corresponds to the supporting hyperplane determined by $\tilde{w}$. It may or may not be the case that $F_{\tilde{w}}$ is a facet of $P_{\tilde{v}}$ (we will find out in the final step); and we know that $z \notin F_{m_1,\dots,m_s}$ if it is not. (This situation is the reason for considering only the interiors of the sets $F_1 + \cdots + F_s$ in the definition of $F_{m_1,\dots,m_s}$.)

Next we determine the face $\tilde{F}$ of $\tilde{P}$ which is induced by the supporting hyperplane orthogonal to $\tilde{w}$. This is done by solving for $i = 1, \dots, s$ the linear program

$$\max\langle w_i, x\rangle \quad \text{s.t.} \quad x \in P_i,$$

where $w_1, \dots, w_s \in \mathbb{R}^n$ such that $\tilde{w} = (w_1^T, \dots, w_s^T)^T$. Note that is is not enough to find *a* solution; we need to find a $\mathcal{V}$- or $\mathcal{H}$-presentation of the set of *all* solutions. But this can be done in polynomial time. So, let $F_1, \dots, F_s$ be the respective solution sets. Then $\tilde{F} = F_1 \times F_2 \times \cdots \times F_s$ is the face of $\tilde{P}$ in question. Now we need to check whether $\dim F_i = m_i$ for all $i = 1, \dots, s$, a task involving just linear algebra, and, if

this is the case, finally, whether $\pi_{\Delta_{\tilde{v}}}$ does not reduce $\dim \tilde{F}$, again a simple task from linear algebra.

Hence we have derived a polynomial-time algorithm for checking membership in $F_{m_1,\dots,m_s}$ which we can now apply to the points x_t used in the proof of the easiness results for zonotopes, and we may proceed as before.

In order to finish the proof of Theorem 7, all that is left to be done is to show that an appropriate choice of the vector $\tilde{v}$ can be made in polynomial time.

The condition on $\tilde{v}$ is satisfied if

$$\bigcap_{i=1}^{s} \big(\text{relint}\,(N(P_i,F_i)) - v_i\big) = \emptyset$$

for all s-tuples $(F_1,F_2,\dots,F_s)$ of faces F_i of P_i such that

$$\dim F_1 + \dim F_2 + \cdots + \dim F_s > n.$$

We will actually produce (in polynomial time) vectors $v_1,\dots,v_s$ such that

$$\bigcap_{i=1}^{s} \big(\text{lin}\,(N(P_i,F_i)) - v_i\big) = \emptyset \tag{2.4}$$

for all s-tuples $(F_1,F_2,\dots,F_s)$ of faces F_i of P_i such that

$$\dim F_1 + \dim F_2 + \cdots + \dim F_s > n.$$

Let $(F_1,F_2,\dots,F_s)$ be such a choice of faces, i.e.,

$$k_1 + \cdots + k_s \geq n+1, \quad \text{where } k_i = \dim F_i \text{ for } i = 1,\dots,s.$$

Suppose that for $i = 1,\dots,s$ the vectors $a_{i,1},\dots,a_{i,n-k_i}$ are facet normals of P_i that span $\text{lin}(N(P_i,F_i))$. Then (2.4) is violated for some choice of $\tilde{v}$, if and only if the following inhomogenous system of linear equations (in the variables x and $\lambda_{i,j}$) is feasible.

$$x + \sum_{j=1}^{n-k_i} \lambda_{i,j} a_{i,j} = v_i \qquad (i = 1,\dots,s). \tag{2.5}$$

Note that this system is overdetermined; it consists of r equations in $n+\sum_{i=1}^{s}(n-k_i) = r + (n - \sum_{i=1}^{s} k_i) \leq r-1$ variables and is, hence, generically infeasible. In order to find a specific vector $\tilde{v}$ of size that is bounded by a polynomial in the input size, which renders *all* such systems infeasible, we have to analyze the condition a bit more carefully, since in general there are doubly exponentially many such systems. Note, however, that the coefficient matrices have the property that all entries are of size that is bounded by a polynomial in the input size. Now suppose $\tilde{v}$ is of the form

$$\tilde{v}_\xi = (v_1^T, v_2^T, \dots, v_s^T)^T = (\xi, \xi^2, \dots, \xi^r)^T \quad \text{for some } \xi > 1.$$

Note that, in general, $\tilde{v}_\xi \notin \Delta^\perp$, but since $\tilde{v}_\xi \notin \Delta$, it is of the form $(y_\xi^T,\dots,y_\xi^T)^T + \tilde{v}'_\xi$ with $\tilde{v}'_\xi \in \Delta^\perp$, whence it suffices to show that for a suitable choice of ξ the system (2.5) is infeasible for $\tilde{v}_\xi$. Now, since (2.5) is overdetermined, it is only feasible if the components of $\tilde{v}_\xi$ satisfy a linear relation with coefficients that come as subdeterminants of (2.5)'s coefficient matrices, whence are bounded in size by an integer polynomial $\pi(\Lambda)$ in the input size Λ, i.e., we have a relation

$$\xi^r + \sum_{i=1}^{r-1} \alpha_i \xi^i = 0 \quad \text{with } |\alpha_1|,\dots,|\alpha_{r-1}| \leq 2^{\pi(\Lambda)}.$$

Hence, with $\xi_0 = 2r2^{\pi(\Lambda)}$ the vector $\tilde{v}_{\xi_0}$ makes all systems (2.5) infeasible.

This completes the proof of the two asserted easiness results. □

2.4. Deterministic methods for approximating mixed volumes. The problem of how well the volume of a well-presented convex body can be approximated in polynomial time was investigated by various authors; see [GK94] for a survey.

For a positive functional ϕ on $\mathcal{K}^n$ (or on appropriate subsets of $\mathcal{K}^n$) and a functional $\lambda : \mathbb{N} \to \mathbb{R}$, a *(relative) λ-approximation* of ϕ is a functional $\hat{\phi}$ defined on the domain of ϕ such that

$$\frac{\phi(K)}{\hat{\phi}(K)} \leq \lambda \quad \text{and} \quad \frac{\hat{\phi}(K)}{\phi(K)} \leq \lambda.$$

(Note that the *relative error* $|(\hat{\phi}(K) - \phi(K))/\phi(K)|$ is only appropriate if one is confronted with small errors since taking $\hat{\phi}(K) = 0$ always gives an estimate with relative error 1.)

When looking for relative estimates for mixed volumes, the first question is if one can efficiently check whether the mixed volume under consideration is greater than zero.

THEOREM 8. *There is a polynomial time algorithm which solves the following problem: given $n, s \in \mathbb{N}$, $m_1, \ldots, m_s \in \mathbb{N}_0$ with $\sum_{i=1}^{s} m_i = n$, and well-presented convex bodies $K_1, \ldots, K_s$, decide whether*

$$V(\overbrace{K_1, \ldots, K_1}^{m_1}, \overbrace{K_2, \ldots, K_2}^{m_2}, \ldots, \overbrace{K_s, \ldots, K_s}^{m_s}) = 0.$$

Proof. For $i = 1, \ldots, s$, let $d_i = \dim(K_i)$, and let $a_{i,0}, \ldots, a_{i,d_i} \in K_i$ such that $\operatorname{aff}(K_i) = \operatorname{aff}\{a_{i,0}, \ldots, a_{i,d_i}\}$. Note that these vectors $a_{i,j}$ are part of the input of the problem. Since our task is clearly translation invariant, we may assume that $a_{1,0} = \cdots = a_{s,0} = 0$, and also that $b_1 = \cdots = b_s = 0$, where b_i is the given "center" of K_i.

Now, for $i = 1, \ldots, s$, let $Z_i = \sum_{j=1}^{d_i} [-1, 1] a_{i,j}$. Then clearly, for some $\rho, R > 0$ (which we do not have to know explicitly),

$$\rho Z_i \subset K_i \subset R Z_i.$$

Hence, by the monotonicity of mixed volumes,

$$V(\overbrace{K_1, \ldots, K_1}^{m_1}, \overbrace{K_2, \ldots, K_2}^{m_2}, \ldots, \overbrace{K_s, \ldots, K_s}^{m_s}) = 0,$$

if and only if

$$V(\overbrace{Z_1, \ldots, Z_1}^{m_1}, \overbrace{Z_2, \ldots, Z_2}^{m_2}, \ldots, \overbrace{Z_s, \ldots, Z_s}^{m_s}) = 0.$$

Using the notation introduced in the previous subsection, let $J_i = \{(i, 1), \ldots, (i, d_i)\}$ for all $i = 1, \ldots, s$, set $J = J_1 \cup \cdots \cup J_s$, set

$$\mathcal{J}_{m_1,\ldots,m_s} = \{I \subset J : |I \cap J_i| = m_i, \text{ for } i = 1, \ldots, s\}$$

and let $A_I = \{a_{i,j} : (i, j) \in I\} \quad \text{for} \quad I \subset J$.

It follows from Proposition 2 that $V(\overbrace{K_1, \ldots, K_1}^{m_1}, \overbrace{K_2, \ldots, K_2}^{m_2}, \ldots, \overbrace{K_s, \ldots, K_s}^{m_s}) \neq 0$ if and only if there is a linear independent subset A_I of A_J which, for $i = 1, \ldots, s$, contains exactly m_i elements from A_{J_i}. This is equivalent to the existence of a common basis for two matroids, the linear matroid and the partition matroid on A_J. The existence of such a common basis can be determined in polynomial time by the matroid intersection algorithm of [Ed70]; see also [GLS88, Theorem 7.5.16]. ∎

Now, assume that $K_1, \dots, K_s$ are well-presented convex bodies and we are longing for relative approximations to

$$V(\overbrace{K_1, \dots, K_1}^{m_1}, \dots, \overbrace{K_s, \dots, K_s}^{m_s}),$$

where $m_1 + \cdots + m_s = n$. Using Proposition 4, we easily obtain a $\left(\min\{\rho_n^{n/2}, \nu_n^{n/2}\}\right)$-approximation of $\mathrm{vol}_n(K)$, where

$$\rho_n = n\sqrt{n+1} \quad \text{and} \quad \nu_n = 2(n+1).$$

(In the rest of the paper we will use these abbreviations to emphasize that improvements in Proposition 4 (in general or for subclasses of $\mathcal{K}^n$) carry over to our approximation results. For such an improvement for $\mathcal{H}$-polytopes see [KT93], and see [GK94] for a survey.) Note that $\rho_n^{n/2}$ and $\nu_n^{n/2}$ depend only on the dimension n and are independent of the bounds of the in- and circumradii given in the input. On the negative side, it has been shown by [BF86] that for each polynomial-time algorithm which produces a λ-approximation of the volume of well-presented convex bodies there exists a constant c such that $\lambda(n) \geq (\frac{cn}{\log n})^{n/2}$ for all $n \in \mathbb{N}$.

It is clear that we cannot expect anything better for mixed volumes, but can we at least get polynomial-time approximations whose error depends only on the dimension n? Note that the "obvious" approach of approximating the bodies $K_1, \dots, K_s$ by parallelotopes each and then using the mixed volume of the parallelotopes as estimates fails in view of Theorem 4, and Theorem 2 indicates some limits for a similar approach using ellipsoids. The following result, however, gives a positive answer for $s = 2$; the general case is open and posed here as a problem.

THEOREM 9. *Let $m : \mathbb{N}_0 \to \mathbb{N}_0$ with $m(n) \leq n$ for all $n \in \mathbb{N}_0$, and let $\lambda : \mathbb{N} \to \mathbb{R}$ be defined by*

$$\lambda(n) = \min\left\{ \rho_n^{\frac{m(n)}{2}} \nu_n^{\frac{n-m(n)}{2}}, \rho_n^{\frac{n-m(n)}{2}} \nu_n^{\frac{m(n)}{2}} \right\}.$$

Then there is a polynomial-time algorithm which produces a λ-approximation of

$$V(\overbrace{K_1, \dots, K_1}^{n-m(n)}, \overbrace{K_2, \dots, K_2}^{m(n)})$$

for well-presented proper convex bodies K_1, K_2.

Proof. Given K_1 and K_2, let ϕ_1 and ϕ_2 be affine transformations such that

$$\mathbb{B}^n \subset \phi_1(K_1) \subset \rho_n \mathbb{B}^n \quad \text{and} \quad C_n \subset \phi_2(K_2) \subset \nu_n C_n.$$

This implies, with $Z = \phi_1(\phi_2^{-1}(C_n))$ and $m = m(n)$, that

$$\begin{aligned} V(\overbrace{\mathbb{B}^n, \dots, \mathbb{B}^n}^{n-m}, \overbrace{Z, \dots, Z}^{m}) &\leq V(\overbrace{\phi_1(K_1), \dots, \phi_1(K_1)}^{n-m}, \overbrace{\phi_1(K_2), \dots, \phi_1(K_2)}^{m}) \\ &\leq V(\overbrace{\rho_n \mathbb{B}^n, \dots, \rho_n \mathbb{B}^n}^{n-m}, \overbrace{\nu_n Z, \dots, \nu_n Z}^{m}). \end{aligned}$$

Since the common affine transformation ϕ_1 changes the mixed volume only by the absolute value of the corresponding determinant as a factor, we obtain the desired bound by taking the geometric mean of the lower and upper estimates and noticing that the roles of K_1 and K_2 can be interchanged. The polynomiality of the algorithm

follows from Proposition 4 and from the fact that the quermassintegrals of a parallelotope can be approximated to absolute positive rational error ϵ in polynomial time in n and $size(\epsilon)$; see, e.g., [GK94, Theorem 4.4.4]. □

Let us point out that Theorem 9 can be extended to improper sets K_1 and K_2 by first using Theorem 8 to check whether the mixed volume under consideration is 0, and if this is not the case, by applying Theorem 9 to the bodies $K_1 + \epsilon\mathbb{B}^n$ and $K_1 + \epsilon\mathbb{B}^n$ for suitably small positive rational ϵ.

The final result of this subsection is needed as preprocessing for the inductive step in the main algorithm of section 3. It is included here because it is approximative in the sense that it gives an algorithmic solution to the (properly phrased variant of the) question of how well a specific mixed volume $V(\overbrace{K_1,\dots,K_1}^{n-k+1},\overbrace{K_2,\dots,K_2}^{k-1})$ of two bodies approximates the "next" one, $V(\overbrace{K_1,\dots,K_1}^{n-k},\overbrace{K_2,\dots,K_2}^{k})$. First we state a theoretical bound which holds after some preliminary normalizations, then we will show how these assumptions can be satisfied in polynomial time.

LEMMA 3. *Let $K_1, K_2 \in \mathcal{K}^n$, let E be an ellipsoid centered at 0 such that $E \subset K_1 \subset \rho_n E$, and let $v_1,\dots,v_n$ be the semi-axis vectors of E, such that $\|v_1\| \le \cdots \le \|v_n\|$. Further, suppose that $\mathbb{B}^n \subset K_2 \subset \rho_n\mathbb{B}^n$ and that $\|v_m\| = 1$. Then*

$$(n+1)^{-4m+5/2} \le \frac{a_{m-1}}{a_m} \le (n+1)^{4m-3/2},$$

where, for $k = m-1, m$,

$$a_k = V(\overbrace{K_1,\dots,K_1}^{n-k},\overbrace{K_2,\dots,K_2}^{k}).$$

Proof. For $i = 1,\dots,n$, set $w_i = v_i/\|v_i\|$. Further, for $j = 1,\dots,m$, let $U_j = \mathrm{lin}\{v_1,\dots,v_j\}$, let $\pi_j : \mathbb{R}^n \to \mathbb{R}^n$ be the orthogonal projection on $U_j^\perp$, and let V_{U_j} and $V_{U_j^\perp}$ denote the (mixed) volume taken in U_j, $U_j^\perp$ (with respect to the standard j- or $(n-j)$-measure in U_j or $U_j^\perp$), respectively.

Let us begin by giving a simple lower estimate for $V(\overbrace{K_1,\dots,K_1}^{n-k},\overbrace{K_2,\dots,K_2}^{k})$, when $k = m-1, m$. Let

$$Q_k = \mathrm{conv}\{\pm w_1,\dots,\pm w_k\}.$$

Then we obtain, with the aid of Proposition 2,

$$\begin{aligned} V(\overbrace{K_1,\dots,K_1}^{n-k},\overbrace{K_2,\dots,K_2}^{k}) &\ge V(\overbrace{K_1,\dots,K_1}^{n-k},\overbrace{Q_k,\dots,Q_k}^{k}) \\ &= \binom{n}{k}^{-1} V_{U_k^\perp}(\overbrace{\pi_k(K_1),\dots,\pi_k(K_1)}^{n-k}) V_{U_k}(\overbrace{Q_k,\dots,Q_k}^{k}) \\ &= \binom{n}{k}^{-1} \mathrm{vol}_{n-k}(\pi_k(K_1))\mathrm{vol}_k(Q_k) = \frac{2^k}{(n-k+1)\cdot\ldots\cdot n}\mathrm{vol}_{n-k}(\pi_k(K_1)) \\ &\ge \left(\frac{2}{n}\right)^k \mathrm{vol}_{n-k}(\pi_k(K_1)). \end{aligned} \tag{2.6}$$

Next we derive upper bounds. Note, first, that

$$\begin{aligned}\|v_1\|K_2 &\subset \rho_n\|v_1\|\mathbb{B}^n \subset \rho_n E \subset \rho_n K_1;\\ K_1 &\subset \pi_1(K_1) + \rho_n[-1,1]v_1;\\ K_2 &\subset \pi_1(K_2) + \rho_n[-1,1]w_1.\end{aligned}$$

Now, again let $k = m-1, m$. Using the monotonicity of mixed volumes we obtain

$$\begin{aligned}&V(\overbrace{K_1,\dots,K_1}^{n-k},\overbrace{K_2,\dots,K_2}^{k})\\ &\quad\le V(\overbrace{\pi_1(K_1)+\rho_n[-1,1]v_1,\dots,\pi_1(K_1)+\rho_n[-1,1]v_1}^{n-k},\\ &\qquad\qquad \overbrace{\pi_1(K_2)+\rho_n[-1,1]w_1,\dots,\pi_1(K_2)+\rho_n[-1,1]w_1}^{k})\\ &\quad= \sum_{i=0}^{n-k}\sum_{j=0}^{k}\binom{n-k}{i}\binom{k}{j} V(\overbrace{\pi_1(K_1),\dots,\pi_1(K_1)}^{n-k-i},\overbrace{\rho_n[-1,1]v_1,\dots,\rho_n[-1,1]v_1}^{i},\\ &\qquad\qquad \overbrace{\pi_1(K_2),\dots,\pi_1(K_2)}^{k-j},\overbrace{\rho_n[-1,1]w_1,\dots,\rho_n[-1,1]w_1}^{j}).\end{aligned}$$

Proposition 2 then yields the following estimate.

$$\begin{aligned}&V(\overbrace{K_1,\dots,K_1}^{n-k},\overbrace{K_2,\dots,K_2}^{k})\\ &\quad\le (n-k)V(\overbrace{\pi_1(K_1),\dots,\pi_1(K_1)}^{n-k-1},\rho_n[-1,1]v_1,\overbrace{\pi_1(K_2),\dots,\pi_1(K_2)}^{k})\\ &\qquad\qquad + kV(\overbrace{\pi_1(K_1),\dots,\pi_1(K_1)}^{n-k},\overbrace{\pi_1(K_2),\dots,\pi_1(K_2)}^{k-1},\rho_n[-1,1]w_1)\\ &\quad= \frac{2(n-k)\rho_n\|v_1\|}{n}V_{U_1^\perp}(\overbrace{\pi_1(K_1),\dots,\pi_1(K_1)}^{n-k-1},\overbrace{\pi_1(K_2),\dots,\pi_1(K_2)}^{k})\\ &\qquad\qquad + \frac{2k\rho_n}{n}V_{U_1^\perp}(\overbrace{\pi_1(K_1),\dots,\pi_1(K_1)}^{n-k},\overbrace{\pi_1(K_2),\dots,\pi_1(K_2)}^{k-1})\\ &\quad\le \frac{2(n-k)\rho_n^2+2k\rho_n}{n}V_{U_1^\perp}(\overbrace{\pi_1(K_1),\dots,\pi_1(K_1)}^{n-k},\overbrace{\pi_1(K_2),\dots,\pi_1(K_2)}^{k-1})\\ &\quad\le 2\rho_n^2 V_{U_1^\perp}(\overbrace{\pi_1(K_1),\dots,\pi_1(K_1)}^{n-k},\overbrace{\pi_1(K_2),\dots,\pi_1(K_2)}^{k-1}).\end{aligned}$$

The same estimate can now be applied inductively; if we do this $k-1$ times for $k=m$ and k times for $k=m-1$ we obtain

(2.7)

$$\begin{aligned}&V(\overbrace{K_1,\dots,K_1}^{n-m},\overbrace{K_2,\dots,K_2}^{m}) \le (2\rho_n^2)^{m-1}V_{U_{m-1}^\perp}(\overbrace{\pi_{m-1}(K_1),\dots,\pi_{m-1}(K_1)}^{n-m},\pi_{m-1}(K_2))\\ &V(\overbrace{K_1,\dots,K_1}^{n-m+1},\overbrace{K_2,\dots,K_2}^{m-1}) \le (2\rho_n^2)^{m-1}\mathrm{vol}_{n-m+1}(\pi_{m-1}(K_1)).\end{aligned}$$

Now we combine the estimates (2.6) and (2.7) with the fact that

$$\pi_{m-1}(K_2) \subset \rho_n \mathbb{B}^{n-m+1} \subset \rho_n \pi_{m-1}(K_1),$$
$$\pi_{m-1}(K_1) \subset \pi_m(K_1) + \rho_n[-1,1]v_m,$$

and obtain

(2.8)
$$V(\overbrace{K_1,\dots,K_1}^{n-m},\overbrace{K_2,\dots,K_2}^{m}) \le (2\rho_n^2)^{m-1} V_{U_{m-1}^\perp}(\overbrace{\pi_{m-1}(K_1),\dots,\pi_{m-1}(K_1)}^{n-m},\pi_{m-1}(K_2))$$
$$\le (2\rho_n^2)^{m-1}\rho_n \mathrm{vol}_{n-m+1}(\pi_{m-1}(K_1)) \le n^{m-1}\rho_n^{2m-1} V(\overbrace{K_1,\dots,K_1}^{n-m+1},\overbrace{K_2,\dots,K_2}^{m-1})$$

and

(2.9)
$$V(\overbrace{K_1,\dots,K_1}^{n-m},\overbrace{K_2,\dots,K_2}^{m}) \ge \left(\frac{2}{n}\right)^m \mathrm{vol}_{n-m}(\pi_m(K_1))$$
$$\ge \frac{2^{m-1}}{n^m\rho_n}\mathrm{vol}_{n-m+1}(\pi_{m-1}(K_1)) \ge \frac{1}{n^m\rho_n^{2m-1}} V(\overbrace{K_1,\dots,K_1}^{n-m+1},\overbrace{K_2,\dots,K_2}^{m-1}).$$

When ρ_n is replaced by its upper bound $(n+1)^{3/2}$, the estimates (2.8) and (2.9) yield the assertion. □

LEMMA 4. *There is a polynomial-time algorithm which constructs, for given well-presented proper convex bodies K_1, K_2 of $\mathbb{R}^n$ and a given $m \in \{1,\dots,n\}$, an affine transformation ϕ and a rational number $\kappa > 0$ such that*

$$1 \le \frac{a'_{m-1}}{a'_m} \le (n+1)^{8m},$$

where, for $k = m-1, m$,

$$a'_k = V(\overbrace{K'_1,\dots,K'_1}^{n-k},\overbrace{K'_2,\dots,K'_2}^{k})$$

is the corresponding mixed volume of the transformed bodies $K'_1 = \kappa\phi(K_1)$ and $K'_2 = \phi(K_2)$.

Proof. Proposition 4 allows us to construct affine transformations ϕ and $\hat{\phi}$ such that

$$\mathbb{B}^n \subset \phi(K_2) \subset \rho_n\mathbb{B}^n \quad \text{and} \quad \mathbb{B}^n \subset \hat{\phi}(\phi(K_1)) \subset \rho_n\mathbb{B}^n.$$

So, let us assume for simplicity of notation that, already,

$$E \subset K_1 \subset \rho_n E \quad \text{and} \quad \mathbb{B}^n \subset K_2 \subset \rho_n\mathbb{B}^n,$$

where $E = A^{-1}\mathbb{B}^n$ for a nonsingular matrix A whose entries are bounded in size by a polynomial in the input size. Now, using the multilinearity of the mixed volumes, Lemma 3 implies

$$(n+1)^{-4m+5/2}\|v_m\| \le \frac{a_{m-1}}{a_m} \le (n+1)^{4m-3/2}\|v_m\|.$$

The problem of computing $\|v_m\|$ is essentially the task of computing the eigenvalues of A^TA, and this can be done in time that is polynomial in the input data and in the binary size of the required precision ϵ. (A conceptually simple way to find the largest eigenvalue of a positive definite matrix A is to perform a binary search on $A - \lambda I$ (with respect to a parameter λ) using the criterion for positive definiteness that the determinants of the $k \times k$ submatrices of the first k rows and columns are positive. The rest is then standard fare in linear algebra.) However, all these quantities are only available up to a polynomially bounded precision. So suppose that ν is a positive rational such that $|\nu - \|v_m\|| \le \epsilon$. Then we obtain

$$(n+1)^{-4m+5/2}(\nu - \epsilon) \le \frac{a_{m-1}}{a_m} \le (n+1)^{4m-3/2}(\nu + \epsilon),$$

whence, with a sufficiently small (but polynomially bounded) positive ϵ,

$$(n+1)^{-4m}\nu \le \frac{a_{m-1}}{a_m} \le (n+1)^{4m}\nu.$$

So, if we rescale K_2 by a factor $(n+1)^{4m}/\nu$, we obtain the asserted inequality. □

3. Randomized algorithms. In this section, we give a randomized algorithm for computing relative approximations of certain mixed volumes of well-presented convex bodies to relative error ϵ whose running time is polynomial in $1/\epsilon$ and the size of the input. We begin with the case of two bodies K_1 and K_2. Our algorithm uses the polynomial-time randomized volume algorithm of Proposition 1 to obtain relative estimates of the values of the polynomial

$$p(x) = \mathrm{vol}_n(K_1 + xK_2) = \sum_{j=0}^{n} c_j x^j = \sum_{j=0}^{n} \binom{n}{j} a_j x^j$$
$$= \sum_{j=0}^{n} \binom{n}{j} V(\overbrace{K_1, \ldots, K_1}^{n-j}, \overbrace{K_2, \ldots, K_2}^{j}) x^j$$

at certain interpolation points. After deriving a basic estimate in subsection 3.1 and showing that the general case of possibly improper convex bodies can be reduced to the case of all bodies in question being proper, we describe a randomized algorithm in subsection 3.2 that computes approximations $\hat{a}_m$ of the mixed volumes a_m of two proper convex bodies recursively. The scaling of Lemma 4 is used as a preprocessing step; it gives a first rough estimate for a_m. The first part of the algorithm uses a search procedure to produce an approximation of the ratio a_{m-1}/a_m to constant error; the second step gives the desired relative approximation of a_m to error ϵ. Subsection 3.2 concludes with the analysis of the complexity of the algorithm, thus establishing Theorem 10 (as stated in the introduction). Subsection 3.3 generalizes the randomized algorithm to more than two convex bodies and proves Theorem 11 (as stated in the introduction).

3.1. A basic estimate and a reduction lemma. The first part of this subsection gives an estimate that is fundamental for the algorithm presented in subsection 3.2.

Let $\xi_0, \ldots, \xi_n$ be (equidistant) interpolation nodes and for $i = 0, \ldots, n$, let $\hat{y}_i$ denote the relative estimate of $y_i = p(\xi_i)$ to error τ. Setting

$$\hat{c}_m = \sum_{k=0}^{n} b_{km} \hat{y}_k,$$

where the b_{km} are again the coefficients of the Lagrange polynomials, (1.5) yields

$$|\hat{c}_m - c_m| \le \tau \max_{i=0,\dots,n} \{q(\xi_i)\} \sum_{k=0}^{n} |b_{km}|.$$

We are, however, interested in a relative approximation, i.e., an estimate of the form

$$|\hat{c}_m - c_m| \le \tau' |c_m|.$$

Using the results of subsection 1.3, it is not hard to see that, in general,

$$\frac{1}{|c_m|} \max_{i=0,\dots,n} \{q(\xi_i)\} \sum_{k=0}^{n} |b_{km}|$$

grows exponentially in n. Unfortunately, the running time of the approximation algorithm of Proposition 1 is polynomial only in the approximation error and not in its *size*. Hence the relative approximations of y_i to error τ that are produced via Proposition 1 cannot be used in this way to give estimates for *all* coefficients in polynomial time. This is the reason for using a small (left upper corner) $r \times r$ submatrix $B^{(r)}$ of the full matrix $B^{(n+1)}$; to allow polynomiality, $(rm)^m$ must be bounded by a polynomial in n.

The following lemma gives a bound for the error $|\hat{c}_m - c_m|$, where the estimate $\hat{c}_m$ is now computed from $B^{(r)}$. The parameters used are all generated later by the algorithm.

LEMMA 5. *Let $m \in \{1, \dots, n\}$, let $r \in \mathbb{N}$ with $r \ge 4m + 7$, let α, γ, and σ be positive reals with $\alpha \ge 1$ such that*

$$\gamma^k a_k \le \begin{cases} \alpha^r \gamma^m \sigma & \text{for } k \le m-1; \\ \gamma^m \sigma & \text{for } k \ge m, \end{cases} \tag{3.1}$$

let $0 < \eta \le 1$, $h = \eta \frac{\gamma}{rn}$, and for $j = 0, \dots, r-1$ let $\xi_j = j \cdot h$. Further, let $\tau > 0$, and for $j = 0, \dots, r-1$ let $\hat{y}_j \in \mathbb{Q}$ such that $|\hat{y}_j - y_j| \le \tau y_j$. Then, taking the estimate

$$\hat{c}_m = h^{-m} \sum_{i=0}^{r-1} b_{im}^{(r)} \hat{y}_i,$$

we have

$$|\hat{c}_m - c_m| \le \frac{\sigma}{\eta^m} \binom{n}{m} \Big((2\alpha)^r e \tau (rm)^m + \frac{2\eta^r}{7!} \Big). \tag{3.2}$$

Proof. It follows from the choice of interpolation nodes and from (3.1) that

$$\begin{aligned} y_j = |p(\xi_j)| \le |p(\xi_r)| &= \sum_{i=0}^{n} c_i (rh)^i = \sum_{i=0}^{n} c_i \eta^i \gamma^i n^{-i} \\ &\le \sigma \gamma^m \alpha^r \sum_{i=0}^{n} n^{-i} \binom{n}{i} \le \sigma \gamma^m \alpha^r \sum_{i=0}^{n} \frac{1}{i!} \le \sigma \gamma^m \alpha^r e. \end{aligned} \tag{3.3}$$

Now let

$$\tilde{c}_m = h^{-m} \sum_{i=0}^{r-1} b_{im}^{(r)} y_i.$$

Since $\binom{n}{m} \geq (\frac{n}{m})^m$, it follows from (1.8) and (3.3) that

$$\begin{aligned} |\tilde{c}_m - \hat{c}_m| &\leq h^{-m}\tau \max\{y_0, \ldots, y_{r-1}\} \cdot \sum_{j=0}^{r-1} |b_{jm}^{(r)}| \ \leq h^{-m}\tau\sigma\gamma^m\alpha^r e \cdot 2^r \\ &\leq (2\alpha)^r \tau \eta^{-m} \sigma e (rm)^m \binom{n}{m}. \end{aligned} \tag{3.4}$$

Now,

$$h^m \tilde{c}_m = \sum_{j=0}^{r-1} b_{jm}^{(r)} y_j = \sum_{i=0}^{n} c_i h^i \sum_{j=0}^{r-1} b_{jm}^{(r)} j^i = c_m h^m + \sum_{i=r}^{n} c_i h^i \sum_{j=0}^{r-1} b_{jm}^{(r)} j^i.$$

Since $r \geq 4m + 7$, whence $7!\ r^{2m} \leq r!$, we obtain, with the aid of (1.9) and (3.1),

$$\begin{aligned} \left|\tilde{c}_m - c_m\right| &= h^{-m} \left| \sum_{i=r}^{n} c_i h^i \sum_{j=0}^{r-1} b_{jm}^{(r)} j^i \right| \leq h^{-m} \left| \sum_{i=r}^{n} c_i h^i r^i \right| \\ &= \eta^{-m}\gamma^{-m}(rn)^m \sum_{i=r}^{n} c_i \gamma^i \eta^i n^{-i} \leq \eta^{r-m}\sigma(rn)^m \sum_{i=r}^{n} \binom{n}{i} n^{-i} \\ &\leq \eta^{r-m}\sigma(rm)^m \binom{n}{m} \sum_{i=r}^{n} \frac{1}{i!} \leq \eta^{r-m}\sigma \binom{n}{m} \frac{2}{7!}. \end{aligned} \tag{3.5}$$

Clearly, (3.4) and (3.5) yield the asserted inequality (3.2). ∎

The next lemma will allow us to reduce the general case of mixed volume computation to the case of proper convex bodies. We use the notation of Theorem 11.

LEMMA 6. *Let $k \in \mathbb{N}$, $k \leq s$, and suppose $\mathcal{A}_k$ is a polynomial-time randomized algorithm that performs the task stated in Theorem* 11 *under the additional assumption that $K_1, \ldots, K_k$ are proper (while $K_{k+1}, \ldots, K_s$ may be improper). Then there exists a polynomial-time randomized algorithm $\mathcal{A}_{k-1}$ that performs the same task under the assumption that $K_1, \ldots, K_{k-1}$ are proper.*

Proof. Let $K_1, \ldots, K_s \in \mathcal{K}^n$ be well presented, let $K_1, \ldots, K_{k-1}$ be proper, and suppose we want to compute the mixed volume

$$v = V(\overbrace{K_1, \ldots, K_1}^{m_1}, \ldots, \overbrace{K_s, \ldots, K_s}^{m_s}).$$

Let us first use Theorem 8 to determine whether $v = 0$. If this is the case, we are done. So suppose that $v \neq 0$. Then, of course, there is a fixed integer polynomial π in the size Λ of the input such that

$$v \geq 2^{-\pi(\Lambda)}.$$

Now, consider for $0 \leq \delta \leq 1$ the mixed volume

$$\begin{aligned} p(\delta) = V(&\overbrace{K_1, \ldots, K_1}^{m_1}, \ldots, \overbrace{K_{k-1}, \ldots, K_{k-1}}^{m_{k-1}}, \overbrace{K_k + \delta\mathbb{B}^n, \ldots, K_k + \delta\mathbb{B}^n}^{m_k}, \\ &\overbrace{K_{k+1}, \ldots, K_{k+1}}^{m_{k+1}}, \ldots, \overbrace{K_s, \ldots, K_s}^{m_s}). \end{aligned}$$

Clearly,

$$p(\delta) = \sum_{i=0}^{m_k} \binom{m_k}{i} p_i \delta^i,$$

where

$$p_i = V(\overbrace{K_1, \ldots, K_1}^{m_1}, \ldots, \overbrace{K_k, \ldots, K_k}^{m_k - i}, \ldots, \overbrace{K_s, \ldots, K_s}^{m_s}, \overbrace{\mathbb{B}^n, \ldots, \mathbb{B}^n}^{i}),$$

and $p_0 = p(0) = v$. Let $R \in \mathbb{N}$ such that $K_1, \ldots, K_s \subset R[-1,1]^n$. Note that such a bound is part of the input. Then we have

$$p(\delta) - v = \delta \sum_{i=1}^{m_k} \binom{m_k}{i} p_i \delta^{i-1} \leq \delta (2R)^n \sum_{i=1}^{m_k} \binom{m_k}{i} \leq \delta (4R)^n.$$

Let $\epsilon \in \mathbb{Q}$ with $0 < \epsilon \leq 1$ be given; set

$$\delta_0 = \frac{\epsilon}{3(4R)^n 2^{\pi(\Lambda)}} \quad \text{and} \quad \tau = \frac{\epsilon}{3}.$$

From the given well-presentation of K_k we can easily derive in polynomial time a well-presentation of $K_k' = K_k + \delta \mathbb{B}^n$; hence we can apply $\mathcal{A}_k$ to the bodies

$$K_1, \ldots, K_{k-1}, K_k', K_{k+1}, \ldots, K_s.$$

We call $\mathcal{A}_k$ with error parameter τ to compute an approximation $\hat{p}$ of $p = p(\delta_0)$ to relative error τ. We take $\hat{v} = \hat{p}$ as an approximation of v, and obtain

$$\begin{aligned} \left| \frac{\hat{v} - v}{v} \right| &= \left| \frac{\hat{p} - v}{p} \right| \left| \frac{p}{v} \right| \leq \left(\frac{|\hat{p} - p|}{p} + \frac{|p - v|}{p} \right) \frac{p}{v} \leq \tau \frac{p}{v} + \frac{p - v}{v} = \tau + (\tau + 1) \frac{p - v}{v} \\ &\leq \tau + 2\delta_0 (4R)^n 2^{\pi(\Lambda)} \leq \frac{\epsilon}{3} + 2\frac{\epsilon}{3} = \epsilon. \end{aligned}$$

Hence $\hat{v}$ is the desired approximation of v to relative error ϵ. □

3.2. Mixed volumes of two proper bodies. We will now describe a randomized algorithm for computing the mixed volumes $a_0, \ldots, a_k$ of two proper convex bodies K_1 and K_2 of $\mathbb{R}^n$ recursively, where $k \leq \psi(n)$, with

$$\psi(n) \leq n \quad \text{and} \quad \psi(n) \log \psi(n) = o(\log n).$$

We use Proposition 1 to compute relative estimates of $\mathrm{vol}_n(K_1 + xK_2)$ for suitable choices of nonnegative rational x. In particular, $a_0 = \mathrm{vol}_n(K_1)$ is already (approximately) available. For the inductive step suppose that, for some $m \in \{1, \ldots, \psi(n)\}$, estimates $\hat{a}_0, \ldots, \hat{a}_{m-1}$ of the mixed volumes $a_0, \ldots, a_{m-1}$, respectively, have already been obtained to relative error, say $\frac{1}{10}$.

By Lemma 4 we may assume that

$$1 \leq q_m = \frac{a_{m-1}}{a_m} \leq (n+1)^{8m},$$

since the transformation underlying Lemma 4 changes a_m by a constant factor and does not affect relative approximation. Clearly,

$$\frac{10}{11} (n+1)^{-8m} \hat{a}_{m-1} \leq a_m \leq \frac{10}{9} \hat{a}_{m-1},$$

and this gives a first $\sqrt{(11/9)}(n+1)^{4m}$ approximation of a_m.

The main routine is divided into two parts. First, we apply a search technique to improve the above approximation of q_m to constant error. Then we run a similar procedure to obtain the required approximation of α_m to relative error ϵ.

1. Search procedure: Set $q_0 = 1$, and let

$$\gamma_0 = \begin{cases} 1 & \text{if } m = 1; \\ \max\left\{\frac{9\hat{a}_{m-2}}{11\hat{a}_{m-1}}, 1\right\} & \text{otherwise.} \end{cases}$$

Now, note that

$$\frac{11\hat{a}_{m-2}}{9\hat{a}_{m-1}} \geq q_{m-1} \geq \frac{9\hat{a}_{m-2}}{11\hat{a}_{m-1}} \quad \text{for } m \geq 2$$

and that the Aleksandrov–Fenchel inequality (1.2) implies that $q_m \geq \max\{q_{m-1}, 1\}$. This yields, for $\gamma = \gamma_0$,

$$q_m \geq \gamma \geq \left(\frac{9}{11}\right)^2 q_{m-1} \geq \frac{2}{3} q_{m-1}. \tag{3.6}$$

In the kth iteration of our search procedure we have $\gamma = \gamma_k$, also satisfying (3.6), hence,

$$\sigma_k := \frac{a_{m-1}}{\gamma_k} \geq \frac{a_{m-1}}{q_m} = a_m.$$

Now the Aleksandrov–Fenchel inequality implies that $\gamma_k \leq q_j$ for all $j \geq m$, hence, inductively,

$$\gamma_k^j a_j \leq \gamma_k^{m-1} a_{m-1} = \gamma_k^m \sigma_k \quad \text{for all } j \geq m.$$

Similarly for $j \leq m-2$, we deduce from $q_{j+1}, \ldots, q_{m-1} \leq \frac{3}{2}\gamma$ that

$$\gamma_k^j a_j \leq \left(\frac{3}{2}\right)^{m-j-1} \gamma_k^{m-1} a_{m-1} \leq \left(\frac{3}{2}\right)^r \gamma_k^m \sigma_k \quad \text{for all } j \leq m-2,$$

whenever $r \geq m$. Let us choose $r \geq 4m + 7$ such that $r = O(\psi)$. Now we apply Lemma 5 with the parameters

$$\gamma = \gamma_k, \quad \sigma = \sigma_k, \quad \alpha = \frac{3}{2}, \quad \eta = 1, \quad \text{and} \quad \tau = \frac{1}{20 \cdot 3^{r+1}(rm)^m}.$$

So, using the volume algorithm of Proposition 1 with interpolation nodes $\xi_j = j \cdot h$ for $j = 0, \ldots, r-1$, where $h = h_k = \frac{\gamma_k}{rn}$, and error bound τ we obtain relative estimates $\hat{y}_j \in \mathbb{Q}$ of $y_j = p(\xi_j)$ that can be used to produce in polynomial time an estimate $\hat{c}_m$ of c_m that satisfies (3.2), whence

$$|\hat{c}_m - c_m| \leq \sigma_k \binom{n}{m} \left(\frac{1}{20} + \frac{2}{7!}\right) \leq \sigma_k \binom{n}{m} \frac{1}{19}.$$

But then we have for

$$s_k = \hat{c}_m / \tbinom{n}{m},$$

$$|s_k - a_m| \leq \frac{1}{19}\sigma_k.$$

Now, if

$$s_k \leq \frac{4 \cdot 10}{9 \cdot 11} \frac{\hat{a}_{m-1}}{\gamma} \leq \frac{4}{9}\sigma_k,$$

then

$$a_m \le \left(\frac{4}{9} + \frac{1}{19}\right)\sigma_k < \frac{\sigma_k}{2},$$

whence $\gamma_k < \frac{q_m}{2}$. So $\gamma_{k+1} = 2\gamma_k$ still satisfies (3.6), and we can repeat the above procedure with γ_{k+1}.

Note that $\sigma_0 \le a_{m-1}$ and $\sigma_{k+1} = \frac{1}{2}\sigma_k$; this implies that after at most $8m\log(n+1)$ iterations the process stops with a $\hat{\gamma} \in \mathbb{Q}$ such that

$$s_k \ge \frac{4\hat{a}_{m-1}}{10\hat{\gamma}} \ge \frac{4}{11}\sigma_k.$$

This implies that

$$\sigma_k \ge a_m \ge \left(\frac{4}{11} - \frac{1}{19}\right)\sigma_k \ge \frac{1}{4}\sigma_k, \tag{3.7}$$

hence $q_m/4 \le \hat{\gamma} \le q_m$. Note that $10\hat{a}_{m-1}/(11\hat{\gamma})$ is already a 4-approximation of a_m.

2. Approximation: Now that we know q_m approximately, we are able to compute $\hat{a}_m$, the desired approximation of a_m to the relative error ϵ. We assume that $0 < \epsilon \le 1$ and choose a positive rational η_0 of size that is bounded by the size of the input such that

$$\eta_0 \le \epsilon^{\frac{1}{r-m}}.$$

As before, we apply Lemma 5, this time with the parameters

$$\gamma = \hat{\gamma}, \quad \sigma = \sigma_k, \quad \alpha = \frac{3}{2}, \quad \eta = \eta_0, \quad \text{and} \quad \tau = \frac{\eta_0^r}{15 \cdot 3^r (rm)^m}.$$

Then we use again the volume algorithm of Proposition 1, now with interpolation nodes $\xi_j = j \cdot h$ for $j = 0, \dots, r-1$, where $h = \frac{\hat{\gamma}}{rn}\eta_0$, and error bound τ, and we obtain relative estimates $\hat{y}_j \in \mathbb{Q}$ of $y_j = p(\xi_j)$ that lead in polynomial time to an estimate $\hat{c}_m$ of c_m that satisfies (3.2), whence

$$|\hat{c}_m - c_m| \le \sigma_k \eta_0^{r-m} \binom{n}{m}\left(\frac{e}{15} + \frac{2}{7!}\right) \le \binom{n}{m}\frac{\epsilon}{4}\sigma_k.$$

In conjunction with (3.7) this yields

$$|\hat{a}_m - a_m| \le \frac{\epsilon}{4}\sigma_k \le \epsilon a_m$$

for $\hat{a}_m = \hat{c}_m / \binom{n}{m}$ as required.

As for its running time, under the stated assumptions on ψ the algorithm uses

$$O(rm\log(n+1)) = o(\log^3 n) \quad \text{calls to the volume estimator}$$

with error, where

$$\frac{1}{\tau} = O(3^r r^{2m}) = n^{o(1)} \quad \text{in the first part,}$$
$$\frac{1}{\tau} = \left(\frac{1}{\epsilon}\right)^{1+o(1)} n^{o(1)} \quad \text{in the final step.}$$

It follows that the algorithm is polynomial. Note that the running time is only marginally worse than the running time of the volume estimator. In fact, suppose that the volume algorithm (after rounding) has complexity

$$O\left(\frac{1}{\epsilon^k} n^l \log\Big(\frac{1}{\beta}\Big)\right).$$

Then the running time of our mixed volume algorithm is bounded by

$$O\left(\frac{1}{\epsilon^{k+o(1)}} n^{l+o(1)} \log\Big(\frac{1}{\beta}\Big)\right).$$

Since by Lemma 6 the initial assumption that K_1 and K_2 are proper is irrelevant, we have completed the proof of Theorem 10.

3.3. Extension to more than two bodies. Now we extend the algorithm to the case of more than two bodies, thus proving Theorem 11. So, let us consider approximating

$$V(\overbrace{K_1,\dots,K_1}^{m_1},\overbrace{K_2,\dots,K_2}^{m_2},\dots,\overbrace{K_{s-1},\dots,K_{s-1}}^{m_{s-1}},\overbrace{K_s,\dots,K_s}^{m_s}),$$

where $\sum_{i=1}^{s} m_i = n$. We may assume again that $K_1,\dots,K_s$ are proper.

Suppose, recursively, we have an approximation procedure whenever only $s-1$ different bodies occur with $3 \le s \le n$. We want to extend it then to all s bodies by considering the polynomial

$$q(x) = \sum_{k=0}^{m} \binom{m}{k} a_k x^k,$$

for $m = m_{s-1} + m_s$, where

$$a_k = V(\overbrace{K_1,\dots,K_1}^{m_1},\dots,\overbrace{K_{s-2},\dots,K_{s-2}}^{m_2},\overbrace{K_{s-1},\dots,K_{s-1}}^{m-k},\overbrace{K_s,\dots,K_s}^{k}),$$

and using estimates of

$$q(\xi_j) = V(\overbrace{K_1,\dots,K_1}^{m_1},\dots,\overbrace{K_{s-2},\dots,K_{s-2}}^{m_2},\overbrace{K_{s-1}+\xi_j K_s,\dots,K_{s-1}+\xi_j K_s}^{m})$$

for suitably chosen interpolation points $\xi_0,\dots,\xi_m$. Note that for the coefficients a_k we still have the Aleksandrov–Fenchel inequality (1.2).

Suppose, now, that ψ satisfies the condition given in Theorem 11, and let $m_1 \ge n - \psi(n)$. Then the degrees of the corresponding mixed volume polynomials are bounded above by $\psi(n)$. This allows us to simplify the procedure by using the whole coefficient matrix. There is, however, one additional difficulty now. We do not have a polynomial-time procedure for obtaining a "good" initial scaling of the sets anymore (as Lemma 4 for $s = 2$) such that the ratio a_{m-1}/a_m of successive coefficients in the polynomial is suitably bounded. We leave it as an open question whether there is an analogue of Lemma 4 for $s \ge 3$ (with a bound that is independent of the well-presentedness parameters ρ_i, R_i). However, the input yields vectors b_i and numbers ρ, R such that for each K_i we have

$$b_i + \rho\mathbb{B}^n \subset K_i \subset b_i + R\mathbb{B}^n,$$

and we may assume without loss of generality that all b_i are 0. Using the monotonicity of mixed volumes we obtain

$$\frac{\rho}{R} \le \frac{a_{m-1}}{a_m} \le \frac{R}{\rho},$$

and this implies that the number of iterations in the binary search part of the procedure is bounded by $O\big(\log(R/\rho)\big)$. Moreover it follows that for each $\xi \ge 0$,

$$\rho(1+\xi)\mathbb{B}^n \subset K_{s-1} + \xi K_s \subset R(1+\xi)\mathbb{B}^n.$$

Hence, we have an additional factor $\log(R/\rho)$ as part of the input to the volume approximator. With these modifications, Theorem 11 is just a corollary to Theorem 10.

4. Related problems and applications. The present section contains various applications of our results to problems in discrete mathematics, combinatorics, computational convexity, algebraic geometry, geometry of numbers, and operations research.

4.1. Counting integer points in integer polytopes. A polytope is called *integer* if all vertices are integer vectors. We denote by $\mathcal{P}^n(\mathbb{Z})$ the set of all integer polyopes of $\mathbb{R}^n$. The lattice point enumerator $G : \mathcal{P}^n(\mathbb{Z}) \to \mathbb{N}$ is counting the number of lattice points of lattice polytopes P, i.e., $G(P) = |P \cap \mathbb{Z}^n|$. The following polynomial expansion of $G(kP)$ is due to [Eh67], [Eh77].

PROPOSITION 5. *There are functionals $G_i : \mathcal{P}^n(\mathbb{Z}) \to \mathbb{N}_0$ such that for every $P \in \mathcal{P}^n(\mathbb{Z})$ and $k \in \mathbb{N}$,*

$$G(kP) = \sum_{i=0}^{n} k^i G_i(P).$$

The polynomial on the right-hand side is often referred to as *Ehrhart-polynomial*; see [St86] for basic facts on this polynomial. In case of lattice zonotopes one can give an explicit formula for the Ehrhart-polynomial; this was used in [St91] to find a generating function for the number of *degree sequences* of simple m-vertex graphs. (In fact, there is a one-to-one correspondence between these degree sequences and the integer points of a suitable zonotope.) The functionals G_i have some interesting properties; see e.g., the survey [GW93]. They may be viewed as "discrete" analogues of the *quermassintegrals*

$$W_i(P) = V(\overbrace{P, \ldots, P}^{n-i}, \overbrace{\mathbb{B}^n, \ldots, \mathbb{B}^n}^{i}).$$

What is particularly important for our purpose is the fact that $G_n(P)$ is just the volume of P. Hence, it follows from Proposition 5 that determining the number of integer points of an integer polytope (that is presented in any of the standard ways) is (at least) as hard as computing its volume. In fact, it is easy to obtain from any standard presentation of an integer polytope P the same kind of presentation for kP of size that is bounded by a polynomial in $size(P)$ and $size(k)$. Hence, if we had a polynomial-time procedure for determining the number of lattice points of an integer polytope, we could run the algorithm for each polytope $0 \cdot P, 1 \cdot P, \ldots, n \cdot P$, and we would then obtain $\mathrm{vol}_n(P) = G_n(P)$ by computing the leading coefficient of the Ehrhard-polynomial, just by solving the corresponding system of linear equations. Hence, we obtain the following $\#\mathbb{P}$-hardness result as a consequence of the hardness results for volume computation of [DF88] (for $\mathcal{V}$- and $\mathcal{H}$-polytopes) and of Theorem 1 (for $\mathcal{S}$-zonotopes).

THEOREM 12. *The problem of evaluating $G(P)$ is $\#\mathbb{P}$-complete for integer $\mathcal{V}$-, integer $\mathcal{H}$-polytopes, and for integer $\mathcal{S}$-zonotopes.*

Let us remark that in fixed dimension $G(P)$ can be computed in a polynomial time, [Ba94]; see also [DK97]. Note, further, that while this task is easy for $\mathcal{V}$-polytopes, deciding whether a given $\mathcal{H}$-polytope P is an integer polytope is co$\mathbb{NP}$-complete; see [PY90]. For a survey of various other results on lattice point enumeration see [GW93].

4.2. Some determinant problems and their relatives. We will proceed by determining the complexity of the following determinant problems (using similar

methods as in the proof of Theorem 1), and then draw some consequences for a problem in computational convexity.

Let κ be an integer constant. Then κ-DETERMINANT is the following decision problem: *given positive integers n, s with $s \geq n$, and an integer $n \times s$-matrix A, is there an $n \times n$-submatrix B of A such that* $\det B = \kappa$?

#(κ-DETERMINANT) asks for the number of different such matrices.

THEOREM 13. *The problem #(κ-DETERMINANT) is #$\mathbb{P}$-complete for any $\kappa \in \mathbb{Z}$; κ-DETERMINANT is $\mathbb{NP}$-complete for $\kappa \neq 0$.*

Proof. Clearly, the first problem is in #$\mathbb{P}$ while the second is in $\mathbb{NP}$. To prove the hardness results, we use reductions from # SUBSET-SUM and SUBSET-SUM, respectively; see the proof of Theorem 1.

Let $(m; \alpha_1, \ldots, \alpha_m, \alpha)$ be an instance of SUBSET-SUM (or, equivalently, of its counting version), and define the matrix A'_δ as in the proof of Theorem 1, but with each α_j replaced by $\beta_j = (|\kappa| + 2)\alpha_j$ for $j = 1, \ldots, m$, $\alpha_{m+1} = -\alpha$ replaced by $\beta_{m+1} = -(|\kappa| + 2)\alpha + 1$, and $\delta = \kappa - 1$. It follows readily that for each maximal square submatrix B_I of A'_δ whose determinant does not depend on δ we have

$$\det B_I \in \{0, \pm\beta_1, \ldots, \pm\beta_m, \pm\beta_{m+1}\}.$$

In particular, $\det B_I \neq \kappa$, unless $\kappa = 0$.

Now, suppose $\kappa \neq 0$. Recall from the proof of Theorem 1 that in the remaining cases the index sets I of the matrices B_I in the determinantal expansion of $\mathrm{vol}_n(Z_\delta)$ are in one-to-one correspondence with the subsets J of $\{1, \ldots, m+1\}$ via

$$j \in J \Longleftrightarrow 2j - 1 \in I.$$

Further, it is easy to see that $\det B_I = \kappa$ implies $m+1 \in J$, and, hence, $\det B_I = \kappa$ if and only $J \setminus \{m+1\}$ is a solution of the given instance of SUBSET-SUM. This settles the problem for $\kappa \neq 0$.

Now, let $\kappa = 0$, and let us use the original matrix A_δ of the proof of Theorem 1. Among the $\binom{2m+3}{m+2} - 2^{m+1}$ subdeterminants for which $\det B_I$ is independent of δ, we have for each $i = 1, \ldots, m+1$ exactly $2^m + m2^{m-1}$ subsets I of $\{1, \ldots, 2m+3\}$ of cardinality $m+2$ such that $|\det B_I| = |\alpha_i|$, and all other cases give $\det B_I = 0$. Hence, with the choice of $\delta = 0$, the number of singular $(m+2) \times (m+2)$ submatrices would allow us to compute the number of subsets $J \subset \{1, \ldots, m+1\}$ for which $\sum_{i \in I} \alpha_i = 0$, and this is the number of solutions of #SUBSET-SUM. □

Let us point out that the (seemingly) similar problem of finding an $n \times n$ submatrix B of maximal determinant of a given $n \times m$ matrix A with entries in $\{0, \pm 1\}$ is also $\mathbb{NP}$-hard even for quite small classes of such matrices A; [see GKL95, Theorem 5.2]. Clearly, this problem is closely related to the problem of finding a largest (with respect to the volume) n-simplex in a $\mathcal{V}$-polytope, while the $\mathbb{NP}$-hardness result of Theorem 13 implies that, given a $\mathcal{V}$-polytope P and a positive integer κ, finding an n-simplex S with vertices at vertices of P and $\mathrm{vol}_n(S) = \kappa$ is $\mathbb{NP}$-complete.

We remark that Theorem 13 stops short of proving that SINGULAR-SUBMATRIX, the case $\kappa = 0$ of κ-DETERMINANT, is also $\mathbb{NP}$-hard. To extend Theorem 13 to SINGULAR-SUBMATRIX would be interesting since, in a geometric context, the existence of singular submatrices corresponds to configurations which are not in general position. However, general position assumptions are made frequently and it would be useful to know whether these assumptions can be checked efficiently; see [CKM82] for some related results.

4.3. Volume of zonotopes and mixture management. The following mixture management problem from the oil industry is studied in [GV89]. A seller has

a stock of m bins which contain a mixture of chemical substances. Suppose that for $i \in \{1,\dots,m\}$ and $j \in \{1,\dots,n\}$, the ith bin contains a nonnegative rational z_{ij} of volume units of chemical j. To satisfy the customer's demand of a special mixture of b_j volume units of each chemical j, the seller takes a proportion λ_i with $0 \le \lambda_i \le 1$ of volume from each container such that

$$b_j = \sum_{i=1}^{m} \lambda_i z_{ij} \quad \text{for all } j \in \{1,\dots,n\}.$$

Typically, the mixtures in each bin come with associated costs, and a linear programming approach is used to satisfy the customer's demand at minimum total cost. It is pointed out in [GV89], however, that this is not a reasonable optimality criterium if all bins have (approximately) the same costs, and this is the case for particular applications in the oil industry.

Therefore [GV89] suggest the following approach. Clearly, the zonotope

$$Z = \sum_{i=1}^{m} [0,1] z_i, \quad \text{where } z_i = (z_{i1},\dots,z_{in})^T \text{ for } i = 1,\dots,m,$$

describes all possible demands the seller can satisfy. Now, typically, there are many possibilities to satisfy a demand $b = (b_1,\dots,b_m)^T$, and the question is how to do it in such a way that "the widest possible variety of possible future demands" can still be satisfied. More precisely, [GV89] suggests choosing for each $b \in \mathbb{Q}^m$ a vector

$$l = \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_m \end{pmatrix} \in L(b) := \left\{ \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_m \end{pmatrix} \in [0,1]^m : b = \sum_{i=1}^{m} \lambda_i z_i \right\}$$

such that after taking λ_i volume proportions from bin i, respectively, the set

$$Z(l) := \sum_{i=1}^{m} [0,1](1-\lambda_i) z_i$$

of the remaining possible mixtures has maximal volume. The volume as objective functions is justified by the fact that if the seller has no information about the future demands, it is reasonable to assume that the future demand is uniformly distributed.

Note that the function $f(l) := \mathrm{vol}_n(Z(l))$ is a homogeneous polynomial in the $(1-\lambda_i)$'s and (due to the *Brunn–Minkowski theorem*) has nice analytic properties. But as we have seen in Theorem 1, evaluating $f(\lambda)$ is $\#\mathbb{P}$-hard. This means that, while intriguing, this approach is not practical for large numbers of bins. However, Proposition 1 suggests that a randomized variant of the algorithm presented in [GV89] may be worth considering.

4.4. Two applications of mixed volumes in combinatorics. Two interesting applications in combinatorics can be found in [St81].

For the first, suppose that $\mathcal{M}$ is a unimodular matroid of rank n with representation $v_1, v_2, \dots, v_m \in \mathbb{R}^n$ over the reals; see [We76], [Wh87]. Let $S_1,\dots,S_s$ be a partition of $\{1,2,\dots,m\}$ into proper subsets, and let $t_1,\dots,t_s$ be nonnegative integers such that $\sum_{i=1}^{s} t_i = n$. Then the number of bases of $\mathcal{M}$ with t_i elements in S_i equals

$$\binom{n}{t_1,\dots,t_s} V(\overbrace{Z_1,\dots,Z_1}^{t_1},\dots,\overbrace{Z_s,\dots,Z_s}^{t_s}),$$

where Z_i is the zonotope

$$Z_i = \sum_{j \in S_i} [0,1]v_j.$$

Note that the total number b of bases of $\mathcal{M}$ can be computed easily because the corresponding matrix A with columns $v_1, \ldots, v_m$ is unimodular and hence, by (2.1),

$$b = \sum_{I \in \mathcal{J}} |\det B_I| = \sum_{I \in \mathcal{J}} (\det B_I)^2 = \det(AA^T).$$

As we will see now, this polynomial-time computability is destroyed for mixed volumes of unimodularly generated zonotopes (unless $\#\mathbb{P} = \mathbb{P}$).

THEOREM 14. *The following problem is $\#\mathbb{P}$-hard:*

Instance: $n, s \in \mathbb{N}$ *and* $m_1, \ldots, m_s \in \mathbb{N}$ *such that* $\sum_{i=1}^{s} m_i = n$, $\mathcal{S}$*-zonotopes* $Z_i = (n, s_i; c_i; z_{i,1}, \ldots, z_{i,s_i})$, *for* $i = 1, \ldots, s$ *such that the* $(n \times r)$*-matrix* A *with columns* $z_{i,j}$ *is unimodular, where* $r = \sum_{i=1}^{s} s_i$.

Task: *Compute the mixed volume*

$$V(\overbrace{Z_1, \ldots, Z_1}^{m_1}, \overbrace{Z_2, \ldots, Z_2}^{m_2}, \ldots, \overbrace{Z_s, \ldots, Z_s}^{m_s}).$$

Proof. We reduce the problem of computing the number of perfect matchings in bipartite graphs to the given problem. For $i = 1, 2$, let $V_i = \{v_{i1}, \ldots, v_{in}\}$, let $V_1 \cap V_2 = \emptyset$, let $E \subset \big\{\{v_{1,j}, v_{2,k}\} : j, k = 1, \ldots, n\big\}$, and set $V = V_1 \cup V_2$. Let us now consider the bipartite graph $G = (V, E)$. Since it can be checked in polynomial time whether G admits a perfect matching, we may assume that the number of perfect matchings of G is not 0. We add an additional vertex $v_{2,n+1}$ to V_2 and the edges $E_{n+1} = \big\{\{v_{1,j}, v_{2,n+1}\} : j = 1, \ldots, n\big\}$ to E and obtain a new bipartite graph $G' = (V', E')$. The node-edge incidence-matrix A' of G' is totally unimodular. It has $2n + 1$ rows but is only of rank $2n$. So we delete the row that corresponds to the new vertex $v_{2,n+1}$, and we obtain a totally unimodular matrix A'' of rank $2n$ with $2n$ rows and $|E'| = |E| + n$ columns. The nonsingular $(2n) \times (2n)$-submatrices B of A'' are in one-to-one correspondence with the spanning trees of G'. (Note that in the totally unimodular case $GF(2)$-singularity is equivalent to $\mathbb{R}$-singularity; see, e.g., [Sc86, section 21.1].) Now, we partition E' into E_{n+1} and the n subsets $E_1, \ldots, E_n$ where E_j is the set of those edges of E which contain $v_{2,j}$ $(j = 1, \ldots, n)$. Further, for $j = 1, \ldots, n+1$, let Z_j be the zonotope that is generated by the column vectors of A'' that correspond to E_j. Then by (2.3), the mixed volume

$$\frac{(2n)!}{n!} V\big(Z_1, Z_2, \ldots, Z_n, \overbrace{Z_{n+1}, \ldots, Z_{n+1}}^{n},\big)$$

is just the number of those spanning trees of G' that contain all edges of E_{n+1} and for $j = 1, \ldots, n$ exactly one edge of E_j.

It is easy to see that the spanning trees with this property are in one-to-one correspondence with the perfect matchings of G. □

For the second application let $P = \{p_1, p_2, \ldots, p_s, q_1, q_2, \ldots, q_{n-s}\}$ be a poset, and suppose that $p_1 < p_2 < \cdots < p_s$. For $j = 1, 2, \ldots, s$ let $N(i_1, i_2, \ldots, i_s)$ denote the number of linear extensions σ of P such that $\sigma(p_j) = i_j$; see [St86]. Then, with $i_0 = 0$ and $i_{s+1} = n + 1$,

$$N(i_1, i_2, \ldots, i_s) = (n-s)! V(\overbrace{K_0, \ldots, K_0}^{i_1 - i_0 - 1}, \ldots, \overbrace{K_s, \ldots, K_s}^{i_{s+1} - i_s - 1}),$$

where $K_j \subset \mathbb{R}^{n-s}$ $(j = 0, 1, \ldots, s)$ are the *order polytopes*, i.e., $x \in K_j$ if and only if for all $i = 1, 2, \ldots, n-s$,

$0 \le x_i \le 1$,
$x_i \le x_k$ if $q_i < q_k$ $(k = 1, 2, \ldots, n-s)$,
$x_i = 0$ if $j > 0$ and $q_i < p_j$,
$x_i = 1$ if $j < s$ and $q_i > p_{j+1}$.

These polytopes reflect the poset "between" p_j and p_{j+1} on the subset $\{q_1, \ldots, q_{n-s}\}$. By the Aleksandrov–Fenchel inequality (applied to in the case $s = 1$) it follows that $N(i)^2 \ge N(i-1)N(i+1)$ for $i = 1, \ldots, n-1$ and, hence, the sequence $N(1), \ldots, N(n)$ is unimodal. Observe that the evaluation of $N(i_1, i_2 \ldots, i_r)$ is $\#\mathbb{P}$-complete even when $s = 0$, [BW92]; in this case, N is the number of linear extensions of the poset. It follows that computing the volume of $\mathcal{H}$-polytopes is $\#\mathbb{P}$-hard in the strong sense.

4.5. An application of mixed volumes in algebraic geometry. Let $S_1, S_2, \ldots, S_n$ be subsets of $\mathbb{Z}^n$, and consider a system $F = (f_1, \ldots, f_n)$ of Laurent polynomials in n variables such that the exponents of the monomials in f_i are in S_i for all $i = 1, \ldots, n$. Suppose, further, that F is *sparse* in that the number of monomials having nonzero coefficients is "small" as compared to the degree of the f_i. To fix the notation, let, for $i = 1, \ldots, n$,

$$f_i(x) = \sum_{q \in S_i} c_q^{(i)} x^q,$$

where $f_i \in \mathbb{C}[x_1, x_1^{-1}, \ldots, x_n, x_n^{-1}]$, and x^q is an abbreviation for $x_1^{q_1} \cdots x_n^{q_n}$; $x = (x_1, \ldots, x_n)$ are the indeterminates and $q = (q_1, \ldots, q_n)$ the exponents. Further, let $\mathbb{C}^* = \mathbb{C} \setminus \{0\}$.

Now, if the coefficients $c_q^{(i)}$ $(q \in S_i)$ are chosen "generically," the number $L(F)$ of distinct common roots of the system F in $(\mathbb{C}^*)^n$ depends only on the *Newton polytopes* $P_i = \operatorname{conv} S_i$ of the polynomials (see [GKZ90]); more precisely,

$$L(F) = n! \cdot V(P_1, P_2, \ldots, P_n). \tag{4.1}$$

Moreover, if F has less then $n!V(P_1, \ldots, P_n)$ distinct roots, there must exist a nonzero integer vector $\alpha = (\alpha_1, \ldots, \alpha_n)$ such that the "homogenized" system

$$F_\alpha = (f_1^\alpha, \ldots, f_n^\alpha),$$

where

$$f_i^\alpha(x) = \sum_{q \in S_i^\alpha} c_q^{(i)} x^q, \qquad S_i^\alpha = \big\{ q \in S_i : \alpha^T q = \min\{\alpha^T q : q \in S_i\} \big\}$$

has a root in $(\mathbb{C}^*)^n$. These results become more intuitive by noting that both sides of (4.1) are invariant under unimodular transformations of the exponent vectors and under translations by integer vectors. (Each translation of a set S_i by a vector $p^{(i)}$ corresponds to a multiplication of f_i with the monomial $x^{p^{(i)}}$.) Observe, further, that the Minkowki sum of the Newton polytopes $P_1, \ldots, P_n$ is the Newton polytope of the product of the corresponding polynomials whence both sides of the equation are also additive in each component.

The above theorem was first proved in [Be75]; see also [BZ88, Chapter 27].

A convex geometric approach (utilizing the above connections) was recently developed for computing the isolated solutions of sparse polynomial systems; see [HS95],

[VG95], and [Ro94]. The mixed volumes are determined by computing a "mixed subdivision" of the P_i using lifting methods similar to those of [Sc94] stated in subsection 2.3. See also [GKZ90] and [GS93] for further results on Newton polytopes and [VC92], [PS93], [CE93], [CR91], [VVC94], [ER94], [EC95], [LRW96], [Ro94], [Ro97], and the papers quoted therein for further results on counting the roots of polynomial systems.

Acknowledgment. We are grateful to Mark Jerrum for providing the proof of Lemma 2.

REFERENCES

[Al37] A.D. Aleksandrov, *On the theory of mixed volumes of convex bodies,* II. *New inequalities between mixed volumes and their applications*, Math. Sb. N.S., 2 (1937), pp. 1205–1238 (in Russian).

[Al38] A.D. Aleksandrov, *On the theory of mixed volumes of convex bodies,* IV. *Mixed discriminants and mixed volumes*, Math. Sb. N.S., 3 (1938), pp. 227–251 (in Russian).

[AS86] E.L. Allgower and P.M. Schmidt, *Computing volumes of polyhedra*, Math. of Comp., 46 (1986), pp. 171–174.

[AK90] D. Applegate and R. Kannan, *Sampling and integration of near log-concave functions*, in Proc. 23rd ACM Symp. on Theory of Computing, ACM, New York, 1990, pp. 156–163.

[BF86] I. Bárány and Z. Füredi, *Computing the volume is difficult*, in Proc. 18th ACM Symp. on Theory of Computing, ACM, New York, 1986, pp. 442–447. Reprinted in Discrete Comput. Geom., 2 (1987), pp. 319–326.

[Ba94] A.I. Barvinok, *A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed*, Math. Oper. Res., 19 (1994), pp. 769–779.

[BZ65] I.S. Berezin and N.P. Zhidkov, *Computing Methods*, Vol. 1, Pergamon Press, Oxford, 1965.

[Be75] D.N Bernshtein, *The number of roots of a system of equations*, Funct. Anal. Appl., 9 (1975), pp. 183–185.

[Be92] U. Betke, *Mixed volumes of polytopes*, Arch. Math., 58 (1992), pp. 388–391.

[BH93] U. Betke and M. Henk, *Approximating the volume of convex bodies*, Discrete Comput. Geom., 10 (1993), pp. 15–21.

[BF34] T. Bonnesen and W. Fenchel, *Theorie der konvexen Körper*, Springer-Verlag, Berlin, 1934 (in German); (reprinted: Chelsea, New York, 1948); *Theory of Convex Bodies*, BCS Associates, Moscow, Idaho, 1987, (in English).

[BW92] G. Brightwell and P. Winkler, *Counting linear extensions is #P-complete*, Order, 8 (1992), pp. 225–242.

[Br86] A. Z. Broder, *How hard is it to marry at random*? *(On the approximation of the permanent)*, in Proc. 18th ACM Symp. on Theory of Computing, ACM, New York, 1986, pp. 50–58.

[BS83] R. A. Brualdi and H. Schneider, *Determinantal identities: Gauss, Schur, Cauchy, Sylvester, Kronecker, Jacobi, Binet, Laplace, Muir, and Cayley*, Linear Algebra Appl., 52–53 (1983), pp. 769–791.

[BZ88] Yu. D. Burago and V. A. Zalgaller, *Geometric Inequalities*, Springer-Verlag, Berlin, 1988.

[CE93] J. Canny and I.Z. Emiris, *An efficient algorithm for the sparse mixed resultant*, in Proc. 10th Intl. Symp. Appl. Algebra, Algebraic Alg., Error-Corr. Codes, Lecture Notes in Comput. Sci. 263, Springer-Verlag, New York, 1993, pp. 89–104.

[CR91] J. Canny and J.M. Rojas, *An optimality condition for determining the exact number of roots of a polynomial system*, in Proc. ACM Intl. Symp. Algebraic Symbolic Comput., Bonn, Germany, ACM, New York, 1991, pp. 96-102.

[CH79] J. Cohen and T. Hickey, *Two algorithms for determining volumes of convex polyhedra*, J. Assoc. Comput. Mach., 26 (1979), pp. 401–414.

[CY77] K.C. Chung and T.H. Yao, *On lattices admitting unique Lagrange interpolation*, SIAM J. Numer. Anal., 14 (1977), pp. 735–743.

[CKM82] R. Chandrasekaran, S.N. Kabadi, and K.G. Murty, *Some NP-complete problems in linear programming*, Oper. Res. Lett., 1 (1982), pp. 101–104.

[DF88] M.E. Dyer and A.M. Frieze, *The complexity of computing the volume of a polyhedron*, SIAM J. Comput., 17 (1988), pp. 967–974.

[DF91] M.E. Dyer and A.M. Frieze, *Computing the volume of convex bodies: a case where randomness provably helps*, in Probabilistic Combinatorics and its Applications, Proceedings of Symposia in Applied Mathematics, Vol. 44, Béla Bollobás, ed., American Mathematical Society, Providence, RI, 1991, pp. 123–169.

[DFK91] M.E. Dyer, A.M. Frieze, and R. Kannan, *A random polynomial time algorithm for approximating the volume of a convex body*, J. Assoc. Comput. Mach., 38 (1991), pp. 1–17.

[DK97] M.E. Dyer and R. Kannan, *On Barvinok's algorithm for counting lattice points in fixed dimension*, Math. Oper. Res., 22 (1997), to appear.

[Ed70] J. Edmonds, *Submodular functions, matroids, and certain polyhedra*, in Combinatorial Structures and their Applications, R. Guy, H. Hanani, N. Sauer, and J. Schönheim, eds., Gordon and Breach, New York, 1970, pp. 69–87.

[Eh67] E. Ehrhart, *Sur un probléme de géometrie diophantienne linéaire*, J. Reine Angew. Math., 226 (1967), pp. 1–29; 227 (1967), pp. 25–49.

[Eh77] E. Ehrhart, *Polynômes arithmétiques et méthode des polyédres en combinatoire*, Birkhäuser, Basel, 1977.

[EC95] I.Z. Emiris and J.F. Canny, *Efficient incremental algorithms for the sparse resultant and the mixed volume*, J. Symbolic Comput., 20 (1995), pp. 117–149.

[ER94] I.Z. Emiris and A. Rege, *Monomial bases and polynomial system solving*, in Proc. 8th ACM Intl. Symp. Algebraic Symbolic Comput. '94, Oxford, UK, ACM, New York, 1994, pp. 114–122.

[Fe36] W. Fenchel, *Inégalités quadratique entre les volumes mixtes des corps convexes*, C.R. Acad. Sci. Paris, 203 (1936), pp. 647–650.

[GJ79] M.R. Garey and D.S. Johnson, *Computers and Intractability*, W.H. Freeman, San Francisco, CA, 1979.

[GKZ90] I.M. Gelfand, M.M. Kapranov, and A.V. Zelevinsky, *Newton polytopes and the classical resultant and discriminant*, Adv. Math., 84 (1990), pp. 237–254.

[GV89] D. Girad and P. Valentin, *Zonotopes and mixture management*, in New Methods in Optimization and their Industrial Uses, J.P. Penot, ed., ISNM87, Birkhäuser, Basel, 1989, pp. 57–71.

[GK94] P. Gritzmann and V. Klee, *On the complexity of some basic problems in computational convexity:* II. *Volume and mixed volumes*, in Polytopes: Abstract, Convex and Computational, T. Bisztriczky, P. McMullen, R. Schneider, and A. Ivic Weiss, eds., Kluwer, Boston, 1994, pp. 373–466.

[GKL95] P. Gritzmann, V. Klee, and D. Larman, *Largest k-simplices in d-polytopes*, Discrete Comput. Geom., 13 (1995), pp. 477–515.

[GS93] P. Gritzmann and B. Sturmfels, *Minkowski addition of polytopes: computational complexity and applications to Gröbner bases*, SIAM J. Discrete Math., 6 (1993), pp. 246–269.

[GW93] P. Gritzmann and J.M. Wills, *Lattice points*, in Handbook of Convex Geometry, P.M. Gruber and J.M. Wills, eds., Elsevier, Amsterdam, 1993, pp. 765–798.

[GLS88] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, Berlin, 1988.

[HS95] B. Huber and B. Sturmfels, *A polyhedral method for solving sparse polynomial systems*, Math. Comput., 64 (1995), pp. 1541–1555.

[JS89] M. R. Jerrum and A. J. Sinclair, *Approximating the permanent*, SIAM J. Comput., 18 (1989), pp. 1149–1178.

[JVV86] M. R. Jerrum, L. G. Valiant, and V.V. Vazirani, *Random generation of combinatorical structures from a uniform distribution*, Theoret. Comput. Sci., 43 (1986), pp. 169–188.

[Jo90] D.S. Johnson, *A catalog of complexity classes*, in Handbook of Theoretical Computer Science. vol. A: Algorithms and Complexity, J. van Leeuwen, ed., Elsevier and M.I.T. Press, Amsterdam and Cambridge, MA, 1990 , pp. 67–161.

[KLS97] R. Kannan, L. Lovász, and M. Simonivits, *Random walks and an $O^*(n^5)$ volume algorithm for convex bodies*, Random Structures Algorithms, 11 (1997), pp. 1–96.

[KKLLL93] N. Karmarkar, R. Karp, R. Lipton, L. Lovász, and M. Luby, *A Monte-Carlo algorithm for estimationg the permanent*, SIAM J. Comput., 22 (1993), pp. 284–293.

[Kh93] L.G. Khachiyan, *Complexity of polytope volume computation*, in New Trends in Discrete and Computational Geometry, J. Pach, ed., Springer-Verlag, Berlin, 1993, pp. 91–101.

[KT93] L.G. Khachiyan and M.J. Todd, *On the complexity of approximating the maximal inscribed ellipsoid for a polytope*, Math. Programming, 61 (1993), pp. 137–159.

[La91] J. Lawrence, *Polytope volume computation*, Math. Comput., 57 (1991), pp. 259–271.

[LRW96] T.Y. Li, J.M. Rojas, and X. Wang, *Counting affine roots of polynomial systems via pointed Newton polytopes*, J. Complexity, 12 (1996), pp. 116–133.

[Lo95] L. Lovász, *Random walks on graphs: A survey*, in Combinatorics: Paul Erdös is 80, Vol. 2, D. Miklós, V. T. Sós and T. Szönyi, eds., Bolyai Soc. Math. Stud. 2, János Bolyai Math. Soc., Budapest, 1995, pp. 353–397.

[LS90] L. Lovász and M. Simonivits, *The mixing rate of Markov chains, an isoperimetric inequality and computing the volume*, in Proc. IEEE 31st Annual Symp. Found. Comput. Sci., IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 364–355.

[LS93] L. Lovász and M. Simonivits, *Random walks in a convex body and an improved volume algorithm*, Random Structures Algorithms, 4 (1993), pp. 359–412.

[Mc70] P. McMullen, *The maximum number of faces of a convex polytope*, Mathematika, 17 (1970), pp. 179–184.

[MM85] G. Miel and R. Mooney, *On the condition number of Lagrangian numerical differentiation*, Appl. Math. Comput., 16 (1985), pp. 241–252.

[Mi11] H. Minkowski, *Theorie der konvexen Körper, insbesondere Begründung ihres Oberflächenbegriffs*, in Collected Works Vol. II, Leipzig, Berlin, 1911, pp. 131–229.

[Mo89] H. L. Montgomery, *Computing the volume of a zonotope*, Amer. Math. Monthly, 96 (1989), p. 431.

[Ol86] C. Olmsted, *Two formulas for the general multivariate polynomial which interpolates a regular grid on a simplex*, Math. Comput., 47 (1986), pp. 275–284.

[PY90] C.H. Papadimitriou and M. Yannakakis, *On recognizing integer polyhedra*, Combinatorica, 10 (1990), pp. 107–109.

[PS93] P. Pedersen and B. Sturmfels, *Product formulas for sparse resultants*, Math. Z., 214 (1993), pp. 377–396.

[Ri75] T.J. Rivlin, *Optimally stable Lagrangian numerical differentiation*, SIAM J. Numer. Anal., 12 (1975), pp. 712–725.

[Ri90] T.J. Rivlin, *Chebyshev Polynomials. From Approximation Theory to Algebra and Number Theory*, 2nd ed., John Wiley, New York, 1990.

[Ro94] J.M. Rojas, *A convex geometric approach to counting the roots of a polynomial system*, Theoret. Comput. Sci., 133 (1994), pp. 105–140.

[Ro97] J.M. Rojas, *Toric intersection for affine root counting*, J. Pure Appl. Math., 133 (1997), to appear.

[Sa74] H.E. Salzer, *Some problems in optimally stable Lagrangian differentiation*, Math. Comput., 28 (1974), pp. 1105–1115.

[Sa93] J.R. Sangwine-Yager, *Mixed volumes*, in Handbook of Convex Geometry, P.M. Gruber and J.M. Wills, eds., Elsevier, Amsterdam, 1993, pp. 43–72.

[Sc93] R. Schneider, *Convex Bodies: The Brunn-Minkowski Theory*, Encyclopedia of Mathematics and its Applications, Vol. 44, Cambridge University Press, Cambridge, MA, 1993.

[Sc94] R. Schneider, *Polytopes and Brunn-Minkowski theory*, in Polytopes: Abstract, Convex and Computational, T. Bisztriczky, P. McMullen, R. Schneider, and A. Ivic Weiss, eds., Kluwer, Boston, 1994, pp. 273–300.

[Sc86] A. Schrijver, *Linear and Integer Programming*, Wiley-Interscience, New York, 1986.

[Sh74] G.C. Shephard, *Combinatorial properties of associated zonotopes*, Canad. J. Math., 26 (1974), pp. 302–321.

[SJ89] A. Sinclair and M. Jerrum, *Approximate counting, uniform generation and rapidly mixing Markov chains*, Inform. Comput., 82 (1989), pp. 93–133.

[St81] R.M. Stanley, *Two combinatorial applications of the Aleksandrov-Fenchel inequalities*, J. Combin. Theory Ser. A, 17 (1981), pp. 56–65.

[St86] R.M. Stanley, *Enumerative Combinatorics*, Vol. 1, Wadsworth & Brooks/Cole, Pacific Grove, CA, 1986.

[St91] R. Stanley, *A zonotope associated with graphical degree sequences*, in Applied Geometry and Discrete Mathematics: The Victor Klee Festschrift, P. Gritzmann and B. Sturmfels, eds., Amer. Math. Soc. and Assoc. Comput. Mach., Providence, RI, 1991, pp. 555–570.

[Va77] L.G. Valiant, *The complexity of computing the permanent*, Theoret. Comput. Sci., 8 (1977), pp. 189–201.

[Va79] L.G. Valiant, *The complexity of enumeration and reliability problems*, SIAM J. Comput., 8 (1979), pp. 410–421.

[VC92] J. Verschelde and R. Cools, *Nonlinear reduction for solving deficient polynomial systems by continuation methods*, Numer. Math., 63 (1992), pp. 263–282.

[VG95] J. Verschelde and K. Gatermann, *Symmetric Newton polytopes for solving sparse polynomial systems*, Adv. Appl. Math, 16 (1995), pp. 95–127.

[VVC94] J. Verschelde, P. Verlinden, and R. Cools, *Homotopies exploiting Newton polytopes for solving sparse polynomial systems*, SIAM J. Numer. Anal., 31 (1994), pp. 915–930.

[We76] D.J.A. Welsh, *Matroid Theory*, Academic Press, London, 1976.

[Wh87] N. White, *Unimodular matroids*, in Combinatorial Geometries, N. White, eds., Cambridge University Press, Cambridge, MA, 1987, pp. 40–52.

SIAM J. COMPUT.
Vol. 27, No. 2, pp. 401–413, April 1998
© 1998 Society for Industrial and Applied Mathematics
004

INTERPOLATING ARITHMETIC READ-ONCE FORMULAS IN PARALLEL*

NADER H. BSHOUTY† AND RICHARD CLEVE†

Abstract. A formula is read-once if each variable appears in it at most once. An arithmetic formula is one in which the operations are addition, subtraction, multiplication, and division (and constants are allowed). We present a randomized (Las Vegas) parallel algorithm for the exact interpolation of arithmetic read-once formulas over sufficiently large fields. More specifically, for n-variable read-once formulas and fields of size at least $3(n^2+3n-2)$, our algorithm runs in $O(\log^2 n)$ parallel steps using $O(n^4)$ processors (where the field operations are charged unit cost). This complements some results from [N.H. Bshouty and R. Cleve, *Proc. 33rd Annual Symposium on the Foundations of Computer Science*, IEEE Computer Science Press, Los Alamitos, CA, 1992, pp. 24–27] which imply that other classes of read-once formulas *cannot* be interpolated—or even learned with membership and equivalence queries—in polylogarithmic time with polynomially many processors (even though they can be learned sequentially in polynomial time). These classes include boolean read-once formulas and arithmetic read-once formulas over fields of size $o(n/\log n)$ (for n variable read-once formulas).

Key words. learning theory, parallel algorithm, read-once formula

AMS subject classifications. 41A05, 41A20, 68Q20, 68T05

PII. S009753979528812X

1. Introduction.

The problem of *interpolating* a formula (from some class C) is the problem of exactly identifying the formula from queries to the assignment (membership) oracle. The interpolation algorithm queries the oracle with an assignment a and the oracle returns the value of the function at a.

There are a number of classes of arithmetic formulas that can be interpolated sequentially in polynomial time as well as in parallel in polylogarithmic time (with polynomially many processors). These include sparse polynomials and sparse rational functions ([BT88, BT90, GKS90b, GrKS88, RB89, GKS90b, SS93, M91]).

A formula over a variable set V is *read-once* if each variable appears at most once in it. An *arithmetic read-once* formula over a field $\mathcal{K}$ is a read-once formula over the basic operations of the field $\mathcal{K}$; addition, subtraction, multiplication, division, and constants are also permitted in the formula. The *size* of an arithmetic formula is the number of instances of variables (i.e., leaves) in it.

Bshouty, Hancock, and Hellerstein [BHH92] present a randomized sequential polynomial-time algorithm for interpolating arithmetic read-once formulas (AROFs) over sufficiently large fields. Moreover, they show that, for arbitrarily sized fields, arithmetic read-once formulas can be *learned* using equivalence queries in addition to membership queries. Other works on AROF and boolean read-once formulas can be found in [AHK89, B2H92, BHHK91, GKS90a, Han90, and HH91].

The question of whether arithmetic read-once formulas can be interpolated (or learned) quickly in parallel depends on the size of the underlying field. It is shown in [BC92] that for arithmetic read-once formulas over fields with $o(n/\log n)$ elements there is *no* polylogarithmic-time algorithm that uses polynomially many processors

*Received by the editors June 13, 1995; accepted for publication (in revised form) January 23, 1996. This research was supported in part by NSERC of Canada.
http://www.siam.org/journals/sicomp/27-2/28812.html
†Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4 (bshouty@cpsc.ucalgary.ca, cleve@cpsc.ucalgary.ca).

(for interpolating as well as learning with membership and equivalence queries [A87], [L88]). Also, a similar negative result holds for boolean read-once formulas.

We present a (Las Vegas) parallel algorithm for the exact interpolation of arithmetic read-once formulas over sufficiently large fields. For fields of size at least $3(n^2 + 3n - 2)$, the algorithm runs in $O(\log^2 n)$ parallel steps using $O(n^4)$ processors (where the field operations are charged unit cost). This complements some results from [BC92] which imply that other classes of read-once formulas *cannot* be interpolated—or even learned with membership and equivalence queries—in polylogarithmic time with polynomially many processors (even though they can be learned sequentially in polynomial time). These classes include boolean read-once formulas and arithmetic read-once formulas over fields of size $o(n/\log n)$ (for n variable read-once formulas).

If the "obvious" parallelizations are made to the interpolating algorithm in [BHH92] (i.e., parallelizations of independent parts of the computation) one obtains a parallel running time that is $\Theta(d)$, where d is the depth of the target formula. Since, in general, d can be as large as $\Theta(n)$, this does not result in significant speedup. Our parallel algorithm uses some techniques from the sequential algorithm of [BHH92] as well as some new techniques that enable nonlocal features of the AROF to be determined in polylogarithmic time.

The parallel algorithm can be implemented on an oracle *parallel random access machine* (PRAM). More specifically, it is an *exclusive-read exclusive-write* (EREW) PRAM—which means that processor's accesses to their communal registers are constrained so that no two processors can read from or write to the same register simultaneously. The EREW PRAM initially selects some random input values (uniformly and independently distributed) and then performs $O(n^3)$ membership queries (via its oracle).

2. Identification with queries. The learning criterion we consider is *exact identification*. There is a formula f called the *target formula*, which is a member of a class of formulas C defined over the variable set V. The goal of the learning algorithm is to halt and output a formula h from C that is equivalent to f.

In a *membership query*, the learning algorithm supplies values $(x_1^{(0)}, \ldots, x_n^{(0)})$ for the variables in V, as input to a *membership oracle*, and receives in return the value of $f(x_1^{(0)}, \ldots, x_n^{(0)})$. Let $f|(x \leftarrow x^{(0)})$ denote the projection of f obtained by hardwiring x to the value $x^{(0)}$. An assignment of values to some subset of a read-once formula's variables defines a *projection*, which is the formula obtained by hardwiring those assigned variables to their values in the formula and then rewriting the formula to eliminate constants from the leaves. Note that if f' is a projection of f, it is possible to simulate a membership oracle for f' using a membership oracle for f.

We say that the class C is learnable in polynomial time if there is an algorithm that uses the membership oracle and interpolates any $f \in C$ in polynomial time in the number of variables n and the size of f. We say that C is efficiently learnable in parallel if there is a parallel algorithm that uses the membership oracle and interpolates any $f \in C$ in polylogarithmic time with polynomial number of processors. In the parallel computation p processors can ask p membership queries in one step.

3. Preliminaries. A *formula* is a rooted tree whose leaves are labeled with variables or constants from some domain and whose internal nodes, or *gates*, are labeled with elements from a set of *basis* functions over that domain. A *read-once formula* is a formula for which no variable appears on two different leaves. An *arithmetic*

read-once formula over a field $\mathcal{K}$ is a read-once formula over the basis of addition, subtraction, multiplication, and division of field elements, whose leaves are labeled with variables or constants from $\mathcal{K}$.

In [BHH92] it is shown that a modified basis can be used to represent any arithmetic read-once formula. Let $\mathcal{K}$ be an arbitrary field. The modified basis for arithmetic read-once formulas over $\mathcal{K}$ includes only two nonunary functions, addition ($+$) and multiplication ($\times$). The unary functions in the basis are $(ax+b)/(cx+d)$ for every $a, b, c, d \in \mathcal{K}$ such that $ad - bc \neq 0$. This requirement is to prevent $ax+b$ and $cx+d$ from being identically 0 or differing by just a constant factor. We can also assume that nonconstant formulas over this modified basis do not contain constants in their leaves. We represent such a unary function as f_A, where

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}.$$

The restriction on a, b, c, and d is equivalent to saying the determinant of A (denoted $\det(A)$) is nonzero.

The value of a read-once formula on an assignment to its variables is determined by evaluating the formula from the bottom up. This raises the issue of division by zero. In [BHH92] this problem is handled by defining basis functions over the extended domain $\mathcal{K} \cup \{\infty, \text{ERROR}\}$, where ∞ represents $1/0$ and ERROR represents $0/0$. For the special values, we define our basis function as follows (assume $x \in \mathcal{K} - \{0\}$, $y \in \mathcal{K} \cup \{\infty, \text{ERROR}\}$, and A is as above).

$$\begin{gathered} y + \text{ERROR} = y \times \text{ERROR} = f_A(\text{ERROR}) = \text{ERROR} \\ x + \infty = x \times \infty = \infty \\ 0 + \infty = \infty \times \infty = \infty \\ 0 \times \infty = \infty + \infty = \text{ERROR} \\ f_A(\infty) = \begin{cases} \frac{a}{c} & c \neq 0 \\ \infty & c = 0 \end{cases} \quad \text{and} \quad f_A\left(\tfrac{-d}{c}\right) = \infty \quad \text{if} \quad c \neq 0. \end{gathered}$$

It is shown in [BHH92] that these definitions are designed so that the output of the read-once formula is the same as it would be if the formula were first expanded and simplified to be in the form $p(x_1, \ldots, x_n)/q(x_1, \ldots, x_n)$ for some polynomials p and q, where $\gcd(p, q) = 1$, and then evaluated.

We say that a formula f is *defined* on the variable set V if all variables appearing in f are members of V. Let $V = \{x_1, \ldots, x_n\}$. We say a formula f *depends* on variable x_i if there are values $x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}$, and $x_i^{(1)}$ in $\mathcal{K}$ for which

$$f(x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}) \neq f(x_1^{(0)}, \ldots, x_{i-1}^{(0)}, x_i^{(1)}, x_{i+1}^{(0)}, \ldots, x_n^{(0)})$$

and for which both those values of f are not ERROR. We call such an input vector $v = (x_1^{(0)}, \ldots, x_n^{(0)})$ a *justifying assignment* for x_i.

Between any two gates or leaves α and β in an AROF, the relationships *ancestor*, *descendant*, *parent*, and *child* refer to their relative positions in the rooted tree. Let $\alpha \leq \beta$ denote that α is a descendant of β (or, equivalently, that β is an ancestor of α). Let $\alpha < \beta$ denote that α is a *proper* descendant of β (i.e., $\alpha \leq \beta$ but $\alpha \neq \beta$). For any pair of variables x_i and x_j that appear in a read-once formula, there is a unique node farthest from the root that is an ancestor of both x_i and x_j, called their *lowest common ancestor*, which we write as $\text{lca}(x_i, x_j)$. We shall refer to the *type* of $\text{lca}(x_i, x_j)$ to mean the basis function computed at that gate. We say that a set W

of variables has a common lca if there is a single node that is the lca of every pair of variables in W.

We define the *skeleton* of a formula f to be the tree obtained by deleting any unary gates in f (i.e., the skeleton describes the parenthesization of an expression with the binary operations, but not the actual unary operations or embedded constants).

We now list a basic property of unary functions f_A that is proved in [BHH92].

PROPERTY 3.1.

1. *The function f_A is a bijection from $\mathcal{K} \cup \{\infty\}$ to $\mathcal{K} \cup \{\infty\}$ if and only if* $\det(A) \neq 0$. *Otherwise, f_A is either a constant value from $\mathcal{K} \cup \{\infty, \text{ERROR}\}$ or a constant value from $\mathcal{K} \cup \{\infty\}$, except on one input value on which it is* ERROR.
2. *The functions f_A and $f_{\lambda A}$ are equivalent for any $\lambda \neq 0$.*
3. *Given any three distinct points $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$, and $p_3 = (x_3, y_3)$,*
   (a) *if p_1, p_2, p_3 are on a line then there exists a unique function f_A with $f_A(x) = ax+b$ that satisfies $f_A(x_1) = y_1$, $f_A(x_2) = y_2$, and $f_A(x_3) = y_3$.*
   (b) *if p_1, p_2, p_3 are not on a line then there exists a unique function f_A with* $\det(A) \neq 0$ *that satisfies $f_A(x_1) = y_1$, $f_A(x_2) = y_2$, and $f_A(x_3) = y_3$.*
4. *If functions f_A and f_B are equivalent and* $\det(A), \det(B) \neq 0$, *then there is a constant λ for which $\lambda A = B$.*
5. *The functions $(f_A \circ f_B)$ and f_{AB} are equivalent.*
6. *If* $\det(A) \neq 0$, *functions f_A^{-1} and $f_{A^{-1}}$ are equivalent.*
7. $f_A(\lambda x) = f_{A\left(\begin{smallmatrix}\lambda & 0\\ 0 & 1\end{smallmatrix}\right)}(x)$ *and* $f_A(\lambda + x) = f_{A\left(\begin{smallmatrix}1 & \lambda\\ 0 & 1\end{smallmatrix}\right)}(x)$.
   $\lambda f_A(x) = f_{\left(\begin{smallmatrix}\lambda & 0\\ 0 & 1\end{smallmatrix}\right)A}(x)$ *and* $\lambda + f_A(x) = f_{\left(\begin{smallmatrix}1 & \lambda\\ 0 & 1\end{smallmatrix}\right)A}(x)$.

4. Collapsibility of operations. Whenever two nonunary gates of the same type in an AROF are separated by only a unary gate, it may be possible to collapse them together to a single nonunary gate of the same type with higher arity. For $\star \in \{+, \times\}$, a unary operation f_A is called *$\star$-collapsible* if

$$f_A(x \star y) \star z \equiv f_B(x) \star f_C(y) \star z$$

for some unary operations f_B and f_C. Intuitively, the above property means that if the f_A gate occurs between two nonunary $\star$ gates then the two $\star$ gates can be "collapsed" into a single $\star$ gate of higher arity, provided that new unary gates can be applied to the inputs.

In [BHH92] it is explained that a unary gate f_A is $\times$-collapsible if and only if A is of the form

$$\begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} 0 & a \\ b & 0 \end{pmatrix},$$

and $+$-collapsible if and only if A is of the form

$$\begin{pmatrix} a & b \\ 0 & c \end{pmatrix}.$$

The following are equivalent definitions of $\star$-collapsible that will be used in this paper.

PROPERTY 4.1. *The following are equivalent.*

1. *f_A is $+$-collapsible.*
2. *$f_A(x) = \alpha x + \beta$ for some $\alpha, \beta \in \mathcal{K}$, and $\alpha \neq 0$.*
3. *$f_A(\infty) = \infty$.*

The following are equivalent.

1. f_A *is* $\times$*-collapsible.*
2. $f_A(x) = \alpha x^\beta$ *for some* $\alpha \in \mathcal{K}$, $\beta \in \{1, -1\}$ *and* $\alpha \neq 0$.
3. $\{f_A(\infty), f_A(0)\} = \{0, \infty\}$.

Proof. We prove the property by showing that $1 \Leftrightarrow 2 \Leftrightarrow 3$. If f_A is $+$-collapsible then

$$A = \begin{pmatrix} a & b \\ 0 & c \end{pmatrix},$$

and therefore $f(x) = (a/c)x + (b/c)$. Since A is nonsingular, $a \neq 0$, $c \neq 0$, and $a/c \neq 0$. $2 \Rightarrow 1$ is obvious. If $f_A(x) = \alpha x + \beta$ for some $\alpha, \beta \in \mathcal{K}$ and $\alpha \neq 0$, then $f_A(\infty) = \alpha\infty + \beta = \infty$. If $f_A(\infty) = \infty$ then, since $f_{\left(\begin{smallmatrix} a & b \\ d & c \end{smallmatrix}\right)}(\infty) = a/d = \infty$, we must have $d = 0$.

The result for $\times$-collapsible is left to the reader. □

In [BHH92], a *three-way justifying assignment* is defined as an assignment of constant values to all but three variables in an AROF such that the resulting formula depends on all of the three remaining variables. For the present results, we require assignments that meet additional requirements, which are defined below.

For any two gates α and β with $\alpha < \beta$, define the α–β *path* as the sequence of gate operations along the path in the tree from α to β (including the operations of α to β at the endpoints of the path). Define a *noncollapsing* three-way justifying assignment as a three-way justifying assignment with the following additional property. For the unassigned variables x, y, and z, if $\mathrm{lca}(x, y) < \mathrm{lca}(x, z)$ and all nonunary operations in the $\mathrm{lca}(x, y)$–$\mathrm{lca}(x, z)$ path are of the same type $\star$ (for some $\star \in \{+, \times\}$), then the function that results from the justifying assignment is of the form

$$f_E(f_C(f_A(x) \star f_B(y)) \star f_D(z)),$$

for some unary operations f_A, f_B, f_C, f_D, and f_E, where f_C is not $\star$-collapsible. Intuitively, this means that, after the justifying assignment, the two gates, $\mathrm{lca}(x, y)$ and $\mathrm{lca}(x, z)$, cannot be collapsed—and thus the relationship $\mathrm{lca}(x, y) < \mathrm{lca}(x, z)$ can still be detected in the resulting function.

Now, define a *total* noncollapsing three-way justifying assignment as a *single* assignment of constant values to all variables in an AROF such that, *for any three variables*, if all but those three are assigned to their respective constants then the resulting assignment is noncollapsing three-way justifying.

5. Parallel learning algorithm. In this section, we present a parallel algorithm for learning AROFs. The algorithm has three principal components: finding a total noncollapsing three-way justifying assignment; determining the skeleton of the AROF; and determining the unary gates of the AROF.

The basic idea is to first construct a graph (that will later be referred to as the least common ancestor hierarchy (LCAH) graph) that contains information about the relative positions of the lcas of all pairs of variables. This cannot be obtained quickly in parallel from justifying assignments because of the possibility that some of the important structure of an AROF "collapses" under any given justifying assignment. However, we shall see that any total noncollapsing justifying assignment is sufficient to determine the entire structure of the AROF at once (modulo some polylog processing).

Once the LCAH graph has been constructed, the skeleton of the AROF can be constructed by discarding some of the structure of the LCAH graph (a "garbage

collection" step). This is accomplished using some simple graph algorithms as well as a parallel prefix sum computation (which is NC^1 computable [LF80]).

Finally, once that skeleton is determined, the unary gates can be determined by a recursive tree contraction method (using results from [B74]).

5.1. Finding a total noncollapsing three-way justifying assignment. In [BHH92], it is proven that, for any triple of variables x, y, and z, by drawing random values (independently) from a sufficiently large field and assigning them to the other variables in an AROF, a three-way justifying assignment for those variables is obtained with high probability. In the parallel algorithm, a three-way justifying assignment that is *total noncollapsing* is required. We show that, if the size of the field $\mathcal{K}$ is at least $O(n^2)$ then the same randomized procedure also yields a total noncollapsing three-way justifying assignment with probability at least $\frac{1}{2}$. Therefore, in time $O(1)$ this step can be implemented.

We shall begin with some preliminary lemmas, and then the precise statement that we require will appear in Corollary 5.4.

LEMMA 5.1. *If $g(y,z) = f_B(f_A(y) \star z)$, where f_A is not $\star$-collapsing, then there exists at most one value $z^{(0)}$ for z such that $f_C(y) \equiv g(y, z^{(0)})$ is $\star$-collapsing.*

Proof. Let $\star = +$. If $f_B(f_A(y)+z_0)$ is $+$-collapsible then by Property 4.1 we have

$$f_B(f_A(y) + z_0) = \alpha y + \beta,$$

where $\alpha \in \mathcal{K}\backslash\{0\}$ and $\beta \in \mathcal{K}$. We substitute $y = \infty$ and get

$$f_B(f_A(\infty) + z_0) = \infty.$$

Since f_A is not $+$-collapsible, by Property 4.1 we have $f_A(\infty) = \gamma \neq \infty$. Solving the above system using Property 3.1 we get

$$z_0 = f_{B^{-1}}(\infty) - \gamma.$$

This shows that there is at most one value of z that makes $f_B(f_A(y)+z)$ $+$-collapsible.

Let $\star = \times$. If $f_B(f_A(y)z_0)$ is $\times$-collapsible then by Property 4.1 we have

$$f_B(f_A(y)z_0) = \alpha y^{\beta},$$

where $\alpha \in \mathcal{K}\backslash\{0\}$ and $\beta \in \{+1, -1\}$. We substitute $y = 0, \infty$ and get

$$\{f_B(f_A(0)z_0), f_B(f_A(\infty)z_0)\} = \{0, \infty\}.$$

Since f_A is not $\times$-collapsible, by Property 4.1, we have that either $f_A(0)$ or $f_A(\infty)$ is not in $\{0, \infty\}$. Suppose $f_A(0) \notin \{0, \infty\}$ and suppose $f_B(f_A(0)z_0) = 0$ (the other cases are similar). Solving this gives

$$z_0 = f_{B^{-1}}(0)/f_A(0).$$

This shows that there is at most one value of z that makes $f_B(f_A(y)z)$ $\times$-collapsible. □

LEMMA 5.2. *Let $F(x_1, \ldots, x_n)$ be an AROF with $lca(x_1, x_2) < lca(x_1, x_3)$ and suppose that all nonunary operations in the $lca(x_1, x_2) - lca(x_1, x_3)$ path are of the same type $\star \in \{+, \times\}$. Let $x_4^{(0)}, \ldots, x_n^{(0)}$ be independently, uniformly, and randomly chosen from $S \subseteq \mathcal{K}$, where $|S| = m$. Then the probability that $x_4^{(0)}, \ldots, x_n^{(0)}$ is a noncollapsing three-way justifying assignment is at least $1 - \left(\frac{3n+1}{m}\right)$.*

Proof. Note that $x_4^{(0)}, \ldots, x_n^{(0)}$ is *not* a noncollapsing three-way justifying assignment if and only if it is not a justifying assignment or there exists a path between the lcas of x_1, x_2, and x_3 such that all nonunary operations are of the same type and the path collapses under the assignment. From [BHH92], the probability of the former condition is at most $\frac{2n+4}{m}$. We need to bound the probability of the latter condition.

We have that $F(x_1, \ldots, x_n)$ is of the form

$$E(f_{H_k}(\cdots f_{H_1}(f_{H_0}(A(x_1) \star B(x_2)) \star C_1) \cdots) \star C_k) \star D(x_3)),$$

where $A(x_1), B(x_2), C_1, \ldots, C_k, D(x_3), E(y)$ may depend on variables from $x_4, \ldots, x_n$ in addition to their marked arguments. Let $\bar{A}(x_1), \bar{B}(x_1), \bar{C}_1, \ldots, \bar{C}_k, \bar{D}(x_3), \bar{E}(y)$ denote the above formulas (respectively) with $x_4^{(0)}, \ldots, x_n^{(0)}$ substituted for the variables $x_4, \ldots, x_n$. Also, let $d_1, \ldots, d_k$ denote the degrees of $C_1, \ldots, C_k$ (respectively) as functions of $x_4, \ldots, x_n$. By the assumption that F is in normal form, f_{H_0} is not $\star$-collapsing. Therefore, by Lemma 5.1, there exists at most one value of C_1 for which $f_{H_1}(f_{H_0}(y) \star C_1)$ is $\star$-collapsing. We can bound the probability of this value occurring for C_1. Since the degree of C_1 is d_1, an application of Schwartz's result in [Sch80] implies the probability that this value occurs for C_1 is at most d_1/m.

Similarly, if $f_{H_1}(f_{H_0}(y) \star C_1)$ is not $\star$-collapsing then Lemma 5.1 implies that there exists at most one value of C_2 for which $f_{H_2}(f_{H_1}(f_{H_0}(y) \star \bar{C}_1) \star C_2)$ is $\star$-collapsing, which occurs with probability at most d_2/m, and so on. It follows that the probability that

$$f_{H_k}(f_{H_{k-1}}(\cdots f_{H_1}(f_{H_0}(y) \star \bar{C}_1) \cdots) \star \bar{C}_k)$$

is $\star$-collapsing is at most $(d_1 + \cdots + d_k)\frac{1}{m} \leq \frac{n-3}{m}$.

The result now follows by summing the two bounds. □

THEOREM 5.3. *Let $F(x_1, \ldots, x_n)$ be an AROF over $\mathcal{K}$, and $x_1^{(0)}, \ldots, x_n^{(0)}$ be chosen uniformly from a set $S \subseteq \mathcal{K}$ with $|S| = m$. Then the probability that $x_1^{(0)}, \ldots, x_n^{(0)}$ is a total noncollapsing three-way justifying assignment is at least $1 - \frac{6n^2}{m}$.*

Proof. First, note that from Lemma 5.2 we can immediately infer that if $x_1^{(0)}, \ldots, x_n^{(0)}$ are drawn independently, uniformly, and randomly from $S \subseteq \mathcal{K}$, where $|S| = m$, then the probability that $x_1^{(0)}, \ldots, x_n^{(0)}$ is a noncollapsing three-way justifying assignment is at most $\binom{n}{3}(\frac{n+1}{2m}) = O(\frac{n^4}{m})$.

To obtain a better bound, consider each subformula C_i that is an input to some nonunary gate in the AROF. By results in [BHH92], there are at most two possible values of C_i that will result in some triple of variables with respect to which the assignment is not three-way justifying (the values are 0 and ∞). Thus, as in the proof of Lemma 5.2, the probability of one of these values arising for C_i is at most $\frac{2d}{m}$, where d is the degree of C_i. Also, from Lemma 5.2, there is at most one value of C_i that will result in a collapsing assignment, and the probability of this arising is at most $\frac{d}{m}$. Thus, the probability of one of the two events above arising is at most $\frac{3d}{m}$, and, since $d \leq n$, this is at most $\frac{3n}{m}$.

Since there are at most $2n$ such subformulas C_i, the probability of any one of them attaining one of the above values is at most $\frac{6n^2}{m}$. □

The constant in the proof of Theorem 5.3 can be improved to obtain probability of

$$1 - \frac{\frac{3}{2}(n^2 + 3n - 2)}{m}$$

by using the following observation. Notice that we upper bounded the degree of each subtree by n. In fact, we can upper bound the degree of the leaves (there are n leaves) by degree 1 since they are variables. Then we have $n-1$ other internal subformulas with respective degrees $2 = d_1 \le d_2 \le \cdots \le d_{n-1}$. It is easy to show that $d_i \le i+1$ (simple induction on the number of nodes). Taking all this into account we obtain the above bound.

By setting $m \ge 3(n^2+3n-2)$, we obtain the following.

COROLLARY 5.4. *Let $F(x_1,\ldots,x_n)$ be an AROF over $\mathcal{K}$, and $x_1^{(0)},\ldots,x_n^{(0)}$ be chosen uniformly from a set $S \subseteq \mathcal{K}$ with $|S| = 3(n^2+3n-2)$. Then the probability that $x_1^{(0)},\ldots,x_n^{(0)}$ is a total noncollapsing three-way justifying assignment is least $\frac{1}{2}$.*

This corollary implies that the expected time complexity of finding a total noncollapsing three-way justifying assignment is $O(1)$.

5.2. Determining the skeleton of a read-once formula in parallel. In this section, we assume that a total noncollapsing three-way justifying assignment is given and show how to construct the skeleton with $O(n^3)$ membership queries in one parallel step followed by $O(\log n)$ steps of computation.

First, suppose that, for a triple of variables x, y, and z, we wish to test whether or not $\mathrm{lca}(x,y) < \mathrm{lca}(x,z)$. If $\mathrm{op}(x,y) \ne \mathrm{op}(x,z)$ then this can be accomplished by a direct application of the techniques in [BHH92], using the fact that we have an assignment that is justifying with respect to variables x, y, and z. On the other hand, if $\mathrm{op}(x,y) = \mathrm{op}(x,z)$ then $\mathrm{lca}(x,y) < \mathrm{lca}(x,z)$ could be difficult to detect with a mere justifying assignment because the justifying assignment might collapse the relative structure between these three variables. If all the nonunary operations in the $\mathrm{lca}(x,y)$–$\mathrm{lca}(x,z)$ path are identical then, due to the fact that we have a noncollapsing justifying assignment, we are guaranteed that the substructure between the three variables does not collapse, and we can determine that $\mathrm{lca}(x,y) < \mathrm{lca}(x,z)$ in $O(1)$ time (again by directly applying techniques in [BHH92]). This leaves the case where $\mathrm{op}(x,y) = \mathrm{op}(x,z)$, but the nonunary operations in the $\mathrm{lca}(x,y)$–$\mathrm{lca}(x,z)$ path are *not* all of the same type. In this case, the techniques of [BHH92] might fail to determine that $\mathrm{lca}(x,y) < \mathrm{lca}(x,z)$ and report them as equal. We shall overcome this problem at a later stage in our learning algorithm by making inferences based on hierarchical relationships with other variables. For the time being, we can, in time $O(1)$ with one processor, compute the following.

$$\textbf{DESCENDANT}(x,y,z) = \begin{cases} \text{YES} & \text{if } \mathrm{lca}(x,y) < \mathrm{lca}(x,z) \text{ and } \mathrm{op}(x,y) \ne \mathrm{op}(x,z); \\ \text{YES} & \text{if } \mathrm{lca}(x,y) < \mathrm{lca}(x,z) \text{ and all nonunary operations} \\ & \text{in the } \mathrm{lca}(x,y)\text{–}\mathrm{lca}(x,z) \text{ path are of the same type;} \\ \text{YES or MAYBE} & \text{if } \mathrm{lca}(x,y) < \mathrm{lca}(x,z) \text{ and } \mathrm{op}(x,y) = \mathrm{op}(x,z) \text{ but } \textit{not} \\ & \text{all nonunary operations in the } \mathrm{lca}(x,y)\text{–}\mathrm{lca}(x,z) \\ & \text{path are of the same type;} \\ \text{MAYBE} & \text{otherwise.} \end{cases}$$

Note that if $\textbf{DESCENDANT}(x,y,z) = \text{YES}$ then it must be that $\mathrm{lca}(x,y) < \mathrm{lca}(x,z)$; however, if $\textbf{DESCENDANT}(x,y,z) = \text{MAYBE}$ then it is possible that $\mathrm{lca}(x,y) < \mathrm{lca}(x,z)$, but $\mathrm{op}(x,y) = \mathrm{op}(x,z)$ and the nonunary operations on the $\mathrm{lca}(x,y)$–$\mathrm{lca}(x,z)$ are not of the same type, or that $\mathrm{lca}(x,y) \not< \mathrm{lca}(x,z)$.

To construct the extended skeleton of an AROF, we first construct its LCAH graph, which is defined as follows.

DEFINITION. *The* LCAH graph *of an AROF with n variables consists of $\binom{n}{2}$ vertices, one corresponding to each (unordered) pair of variables. For the distinct*

variables x and y denote the corresponding vertex by xy or, equivalently, yx. Then, for distinct vertices xy and zw, the directed edge $xy \to zw$ is present in the LCAH graph if and only if $lca(x, y) \leq lca(z, w)$.

We shall prove that the following algorithm constructs the LCAH graph of an AROF.

Algorithm CONSTRUCT-LCAH-GRAPH.

1. **in parallel for all distinct variables** x, y, z **do**
   if DESCENDANT(x, y, z) = YES **then**
   insert edges $xy \to xz$ and $xy \to yz$ and $xz \to yz$ and $yz \to xz$
2. **in parallel for all distinct variables** x, y, z, w **do**
   if edges $xy \to xw \to xz$ are present **then**
   insert edge $xy \to xz$
3. **in parallel for all distinct variables** x, y, z **do**
   if no edges between any of xy, xz, yz are present **then**
   insert edges in each direction between every pair of xy, xz, yz
4. **in parallel for all distinct variables** x, y, z, w **do**
   if edges $xy \to xw \to zw$ present **or** edges $xy \to yw \to zw$ present **then**
   insert edge $xy \to zw$

THEOREM 5.5. *Algorithm* **CONSTRUCT-LCAH-GRAPH** *constructs the LCAH graph of an AROF.*

Proof. The proof follows from the following sequence of observations.

(i) For all distinct variables x, y, and z for which $\text{lca}(x, y) < \text{lca}(x, z) = \text{lca}(y, z)$, after executing steps 1 and 2 of the algorithm the appropriate edges pertaining to vertices xy, xz, and yz (namely, $xy \to xz$, $xy \to yz$, $xz \to yz$, and $yz \to xz$) are present.

(ii) For all distinct variables x, y, and z for which $\text{lca}(x, y) = \text{lca}(x, z) = \text{lca}(y, z)$, after executing step 3 of the algorithm the appropriate edges pertaining to vertices xy, xz, and yz (namely, edges in both directions between every pair) are present.

(iii) For all distinct variables x, y, z, and w, after executing step 4 of the algorithm the edge $xy \to zw$ is present if and only if $\text{lca}(x, y) \leq \text{lca}(z, w)$. □

It is straightforward to verify that algorithm **CONSTRUCT-LCAH-GRAPH** can be implemented to run in $O(\log n)$ time on an EREW PRAM with $O(n^4)$ processors. Moreover, the $O(n^3)$ membership queries can be made initially in one parallel step.

In an AROF, each nonunary gate corresponds to a biconnected component (which is a clique) of its LCAH graph. Thus, to transform the LCAH graph into the extended skeleton of the AROF, we simply "compress" each of its biconnected components into a single vertex and then extract the underlying tree structure of this graph (where the underlying tree structure of a graph is the tree whose transitive closure is the graph[1]). This is accomplished using standard graph algorithm techniques, including a parallel prefix sum computation [LF80]. The details follow.

We first designate a "leader" vertex for each biconnected component. We then record the individual variables that are descendants of each nonunary gate and then discard the other nodes in each biconnected component.

The algorithm below selects a leader from each connected component in an LCAH graph. We assume that there is a total ordering $\prec$ on the vertices of the LCAH graph

[1] All edges are directed toward the root.

(for example, the lexicographic ordering on the pair of indices of the two variables corresponding to each vertex).

Algorithm LEADER.

in parallel for all vertices $xy \prec zw$ **do**
if edges $xy \to zw$ **and** $zw \to xy$ are present **then**
mark xy with X.

It is easy to prove the following.

LEMMA 5.6. *After executing algorithm* **LEADER**, *there is precisely one unmarked node (namely, the largest in the* $\prec$ *ordering) in each biconnected component of the LCAH graph.*

After selecting a leader from each biconnected component of the LCAH graph, we add n new nodes to this graph that correspond to the n variables. The edge $x \to yz$ is inserted if and only if the variable x is a descendant of $\mathrm{lca}(y, z)$. This is accomplished by the following algorithm.

Algorithm LEAVES.

in parallel for all distinct variables x, y, z, w **do**
insert edge $x \to xy$
if edge $xy \to zw$ is present **then**
insert edge $x \to zw$.

LEMMA 5.7. *After executing algorithm* **LEAVES**, *the edge* $x \to yz$ *is present if and only if variable* x *is a descendant of* $lca(y, z)$.

Both Algorithms **LEADER** and **LEAVES** can be implemented in $O(\log n)$ time with $O(n^4)$ processors.

After these steps, the marked nodes are discarded from the augmented LCAH graph (that contains $\binom{n}{2} + n$ vertices), resulting in a graph with at most $2n - 1$ vertices that is isomorphic to the extended skeleton of the AROF. This discarding is accomplished by a standard technique involving the computation of prefix sums. We first adopt the convention that the order $\prec$ extends to the augmented LCAH graph as $x_1 \prec \cdots \prec x_n$ and $x \prec yz$ for any variables x, y, and z. Then, for each node v, set

$$\varphi(v) = \begin{cases} 1 & \text{if } v \text{ is unmarked} \\ 0 & \text{if } v \text{ is marked,} \end{cases}$$

and compute the prefix sums

$$\sigma(v) = \sum_{u \preceq v} \varphi(u).$$

With algorithms for parallel prefix sum computation [LF80] this can be accomplished in $O(\log(\binom{n}{2} + n)) = O(\log n)$ time with $O(\binom{n}{2} + n) = O(n^2)$ processors.

The function σ is a bijection between the unmarked nodes of the augmented LCAH graph and some $S \subseteq \{1, 2, \ldots, 2n - 1\}$, and $\sigma(x_i) = i$ when $i \in \{1, \ldots, n\}$. The following algorithm uses the values of this function to produce the extended skeleton of the AROF.

Algorithm COMPRESS-AND-PRUNE.

in parallel for all distinct vertices u, v **do**
if vertices u, v are both unmarked

and edge $u \to v$ is in augmented LCAH graph **then**
insert edge $\sigma(u) \to \sigma(v)$ in skeleton graph
in parallel for all distinct $i, j, k \in S$ **do**
if edges $i \to j \to k$ **and** $i \to k$ are in skeleton graph **then**
remove edge $i \to k$ from skeleton graph.

The following is straightforward to prove.

LEMMA 5.8. *The "skeleton" graph that* **COMPRESS-AND-PRUNE** *produces is isomorphic to the extended skeleton of the AROF, where the inputs* $x_1, \ldots, x_n$ *correspond to the vertices* $1, \ldots, n$ *(respectively) of the graph.*

5.3. Determining a read-once formula from its skeleton. Once the skeleton of an AROF is determined, what remains is to determine the constants in its unary gates (note that the nonunary operations are easy to determine using the techniques in [BHH92]). We show how to do this in $O(\log^2 n)$ steps with $O(n \log n)$ processors. The main idea is to find a node that partitions the skeleton into three parts whose sizes are all bounded by half of the size of the skeleton. Then the unary gates are determined on each of the parts (in a recursive manner), and the unary gates required to "assemble" the parts are computed.

The following lemma is an immediate consequence from a result in [B74].

LEMMA 5.9. *For any formula* $F(x_1, \ldots, x_n)$*, there exists a nonunary gate of type* $\star$ *that "evenly" partitions it in the following sense. With a possible relabelling of the indices of the variables,*

$$F(x_1, \ldots, x_n) \equiv G(f_A(f_B(H(x_1, \ldots, x_k)) \star f_C(I(x_{k+1}, \ldots, x_l))), x_{l+1}, \ldots, x_n),$$

and the number of variables in $G(y, x_{l+1}, \ldots, x_n)$*,* $H(x_1, \ldots, x_k)$*, and* $I(x_{k+1}, \ldots, x_l)$ *are all bounded above by* $\lceil \frac{n}{2} \rceil$*.*

A minor technicality in the above lemma is that, since the skeleton is not necessarily a binary tree, it may be necessary to "split" a nonbinary gate into two smaller gates.

It is straightforward to obtain the above decomposition of a skeleton in NC^1. Once this decomposition is obtained, the recursive algorithm for computing the unary gates of the AROF follows from the following lemma.

LEMMA 5.10. *Let* $x_1^{(0)}, \ldots, x_n^{(0)}$ *be a total noncollapsing justifying assignment for the AROF* $F(x_1, \ldots, x_n)$*. If*

$$F(x_1, \ldots, x_n) \equiv G(f_A(f_B(H(x_1, \ldots, x_k)) \star$$

$$f_C(I(x_{k+1}, \ldots, x_l))), x_{l+1}, \ldots, x_n)$$

then
(i) *given the skeleton of* $F(x_1, \ldots, x_n)$ *and the subformulas* $G(y, x_{l+1}, \ldots, x_n)$*,* $H(x_1, \ldots, x_k)$ *and* $I(x_{k+1}, \ldots, x_l)$*, it is possible to determine* A*,* B*, and* C*, and, thus, the entire structure of* $F(x_1, \ldots, x_n)$ *in* $O(\log n)$ *steps with* $O(n \log n)$ *processors.*
(ii) *given the skeleton of* $F(x_1, \ldots, x_n)$*, the problem of determining* $G(y, x_{l+1}, \ldots, x_n)$*,* $H(x_1, \ldots, x_k)$*, and* $I(x_{k+1}, \ldots, x_l)$ *is reducible to the problem of determining a ROF given its skeleton.*

Proof. For part (i), assume that the subformulas $G(y, x_{l+1}, \ldots, x_n)$, $H(x_1, \ldots, x_k)$, and $I(x_{k+1}, \ldots, x_l)$ are given. Since $x_1^{(0)}, \ldots, x_n^{(0)}$ is a justifying assignment, $G(y, x_{l+1}^{(0)},$

$\ldots, x_n^{(0)})$, $H(x_1, x_2^{(0)}, \ldots, x_k^{(0)})$, $I(x_{k+1}, x_{k+2}^{(0)}, \ldots, x_l^{(0)})$ are all nonconstant unary functions, so there exist nonsingular matrices A', B', C' (which are easy to determine in $O(\log n)$ parallel steps) such that

$$f_{A'}(y) \equiv G\left(y, x_{l+1}^{(0)}, \ldots, x_n^{(0)}\right)$$
$$f_{B'}(x_1) \equiv H\left(x_1, x_2^{(0)}, \ldots, x_k^{(0)}\right)$$
$$f_{C'}(x_{k+1}) \equiv I\left(x_{k+1}, x_{k+2}^{(0)}, \ldots, x_l^{(0)}\right).$$

Also,

$$F(x_1, x_2^{(0)}, \ldots, x_k^{(0)}, x_{k+1}, x_{k+2}^{(0)}, \ldots, x_n^{(0)}) \equiv f_{A'}(f_A(f_B(f_{B'}(x_1) \star f_C(f_{C'}(x_{k+1}))))),$$

so the matrices $A' \cdot A$, $B \cdot B'$, $C \cdot C'$ can be determined in $O(1)$ steps, [BHH92]. From this, the matrices A, B, C can be determined.

For part (ii), consider the problem of determining $G(y, x_{l+1}, \ldots, x_n)$. Note that

$$F(y, x_2^{(0)}, \ldots, x_l^{(0)}, x_{l+1}, \ldots, x_n) \equiv G(f_{A''}(y), x_{l+1}, \ldots, x_n),$$

for some nonsingular A''. Therefore, if we fix $x_2, \ldots, x_l$ to $x_2^{(0)}, \ldots, x_l^{(0)}$ then we have a reduction from the problem of determining $G(f_{A''}(y), x_{l+1}, \ldots, x_n)$.

Similarly, we have reductions from the problem of determining $f_{B''}(H(x_1, \ldots, x_k))$ and $f_{C''}(I(x_{k+1}, \ldots, x_l))$ for nonsingular matrices B'' and C''. Since the matrices A'', B'', C'' can be absorbed into the processing of part (i), this is sufficient. □

By recursively applying Lemmas 5.9 and 5.10, we obtain a parallel algorithm to determine an AROF given its skeleton and a total noncollapsing three-way justifying assignment in $O(\log^2 n)$ steps. The processor count for this can be bounded by $O(n \log n)$.

REFERENCES

[A87] D. Angluin, *Queries and concept learning*, Mach. Learning, 2 (1987), p. 319–342.

[AHK89] D. Angluin, L. Hellerstein, and M. Karpinski, *Learning read-once formulas with queries*, J. ACM, 40 (1993), pp. 185–210.

[AK91] D. Angluin and M. Kharitonov, *When won't membership queries help?*, J. Comput. System Sci., 50 (1995), pp. 336–355.

[B74] J. P. Brent, *The parallel evaluation of general arithmetic expressions*, J. ACM, 21 (1974), pp. 201–206.

[BHH92] N. H. Bshouty, T. R. Hancock, and L. Hellerstein, *Learning arithmetic read-once formulas*, SIAM J. Comput., 24 (1995), pp. 706–735.

[BC92] N. H. Bshouty and R. Cleve, *On the exact learning of formulas in parallel*, Proc. of the 33rd Annual Symposium on Foundations of Computer Science, IEEE Computer Science Press, Los Alamitos, CA, 1992, pp. 24–27.

[B2H92] N. H. Bshouty, T. R. Hancock, and L. Hellerstein, *Learning boolean read-once formulas with arbitrary symmetric and constant fan-in gates*, J. Comput. System Sci., 50 (1995), pp. 521–542.

[BGHM93] N. H. Bshouty, S. Goldman, T. Hancock, and S. Matar, *Asking questions to minimize errors*, in Proc. of the Sixth Annual Workshop on Computational Learning Theory, ACM, New York, 1993, pp. 41–50.

[BHHK91] N. H. Bshouty, T. R. Hancock, L. Hellerstein, and M. Karpinski, *An algorithm to learn read-once threshold formulas, and transformation between learning models*, Comput. Complexity, 4 (1994), pp. 37–61.

[BT88] M. Ben-Or and P. Tiwari, *A deterministic algorithm for sparse multivariate polynomial interpolation*, in Proc. of the 20th Annual ACM Symposium on the Theory of Computing, ACM, New York, 1988, pp. 301–309.

[BT90] A. Borodin and P. Tiwari, *On the decidability of sparse univariate polynomial interpolation*, Comput. Complexity, 1 (1991), pp. 67–90.

[GKS90a] S. A. Goldman, M. J. Kearns, and R. E. Schapire, *Exact identification of read-once formulas using fixed points of amplification functions*, SIAM J. Comput., 22 (1993), pp. 705–726.

[GKS88] D. Y. Grigoriev, M. Karpinski, and M. F. Singer, *Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields*, SIAM J. Comput., 19 (1990), pp. 1059–1083.

[GKS90b] D. Y. Grigoriev, M. Karpinski, and M. F. Singer, *Interpolation of sparse rational functions without knowing bounds on the exponent*, in Proc. of the 31st Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 193–202.

[Han90] T. Hancock, *Identifying μ-formula decision trees with queries*, in Proc. of the Third Annual Workshop on Computational Learning Theory, ACM, New York, 1990, pp. 23–37.

[HH91] T. Hancock and L. Hellerstein, *Learning read-once formulas over fields and extended bases*, in Proc. of the Fourth Annual Workshop on Computational Learning Theory, ACM, New York, 1991, pp. 326–336.

[HS80] J. Heintz and C. P. Schnorr, *Testing polynomials that are easy to compute*, in Proc. of the 12th Annual ACM Symposium on the Theory of Computing, ACM, New York, 1980, pp. 262–272.

[LF80] R. E. Ladner and M. J. Fischer, *Parallel prefix computation*, J. ACM, 27 (1980), pp. 831–838.

[L88] N. Littlestone, *Learning quickly when irrelevant attributes abound: A new linear threshold algorithm*, Mach. Learning, 2 (1988), pp. 285–318.

[M91] Y. Mansour, *Randomized approximation and interpolation of sparse polynomials*, SIAM J. Comput., 24 (1995), pp. 357–368.

[MT90] W. Maass and G. Turán, *On the complexity of learning from counterexamples and membership queries*, in Proc. of the 31st Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 203–210.

[RB89] M. R. Roth and G. M. Benedek, *Interpolation and approximation of sparse multivariate polynomials over GF(2)*, SIAM J. Comput., 20 (1991), pp. 291–314.

[SS93] R. E. Schapire and L. M. Sellie, *Learning sparse multivariate polynomials over a field with queries and counterexamples*, J. Comput. Systems Sci., 52 (1996), pp. 201–213.

[Sch80] J. T. Schwartz, *Fast polynomial algorithms for verification of polynomial identities*, J. Assoc. Comput. Mach., 27 (1980), pp. 701–707.

[Val84] L. G. Valiant, *A theory of the learnable*, Comm. ACM, 27 (1984), pp. 1134–1142.

[VL92] J. S. Vitter and J. Lin, *Learning in parallel*, Inform. and Comput., 96 (1992), pp. 179–202.

SIAM J. COMPUT.
Vol. 27, No. 2, pp. 414–422, April 1998
© 1998 Society for Industrial and Applied Mathematics
005

AN ON-LINE ALGORITHM FOR SOME UNIFORM PROCESSOR SCHEDULING*

RONGHENG LI† AND LIJIE SHI‡

Abstract. This paper considers the problem of on-line scheduling a set of independent jobs on m uniform machines $(M_1, M_2, \ldots, M_m)$ in which machine $M_i's$ processing speed is $s_i = 1(i = 1, \ldots, m-1)$ and $s_m = s > 1$. List scheduling [Y. Cho and S. Sahni, *SIAM J. Comput.*, 9 (1980), pp. 91–103] guarantees a worst-case performance of $\frac{3m-1}{m+1}(m \geq 3)$ and $\frac{1+\sqrt{5}}{2}(m = 2)$ for this problem. We prove that this worst-case bound cannot be improved for $m = 2$ and $m = 3$ and for every $m \geq 4$, an algorithm with worst-case performance at most $\frac{3m-1}{m+1} - \varepsilon$ is presented when $s_m = 2$, where ε is a fixed positive number, and then we improve the bound for general $s_m = s > 1$.

Key words. on-line scheduling, list scheduling, uniform machines

AMS subject classifications. 90B35, 90C27

PII. S00975397995279694

1. Introduction.

A uniform machine system consists of $m, m \geq 1$, machines $(M_1, M_2, \ldots, M_m)$. A speed $s_i, s_i \geq 1$ is associated with each machine. In one unit of time M_i can carry out s_i units of processing. A list $(t_1, t_2, \ldots, t_n)$ of n jobs is given to us on-line—that means we get the jobs one by one. Both the total number of jobs that need to be scheduled and the size of the jobs are not previously known. The processing time of job t_i becomes known only when t_{i-1} has already been scheduled. As soon as job t_i appears, it must irrevocably be scheduled. Our goal is to minimize the makespan, i.e., the maximum completion time over all jobs in a schedule. The quality of an algorithm H is measured by its worst-case ratio

$$R^H(m) = \sup_L \{C^H(L)/C^*(L) : L \textit{ is a list of jobs}\},$$

where $C^H(L)$ denotes the makespan produced by the heuristic H on the machines and the list L of jobs and $C^*(L)$ denotes the corresponding makespan in some optimal schedule. List scheduling (LS), which always assigns the current job to the machine that will complete it first, is a simple example of a nonpreemptive on-line algorithm and is always used. When $s_i = 1(i = 1, 2, \ldots, m)$, the machine system is well known as an identical machine system. For an identical machine system, Graham showed in 1969 that $R^{LS}(m) = 2 - 1/m$ [2]. For $m \geq 4$, this bound was improved by Galambos and Woeginger in 1993 [3]. They designed an algorithm called Refined List Scheduling (RLS) and showed that $R^{RLS}(m) \leq 2 - 1/m - \eta_m$, where $\eta_m > 0$ for $m \geq 4$. Bartal et al. [4] made some progress for large numbers of machines by devising an algorithm whose worst-case guarantee is $2 - \frac{1}{7}$ for all m. When $s_i = 1(i = 1, \ldots, m-1)$ and $s_m = s \geq 1$, Cho and Sahni showed in 1980 that $R^{LS}(m, s) \leq 1 + \frac{m-1}{m+s-1} \min\{2, s\} \leq 3 - \frac{4}{m+1}$, and the bound $3 - \frac{4}{m+1}$ is achieved when s=2.

In this paper, for $m \geq 4$ machines we present a heuristic that has a significantly better worst-case performance guarantee than LS when $s_i = 1(i = 1, \ldots, m-1)$ and

*Received by the editors January 9, 1995; accepted for publication January 25, 1996.
http://www.siam.org/journals/sicomp/27-2/27969.html

†Department of Mathematics, Hunan Normal University, Changsha 410081, People's Republic of China (lap@hunnu.edu.cn).

‡RUTCOR, Rutgers University, New Brunswick, NJ 08903 (lijie@rutcor.rutgers.com).

$s_m = 2$ and then show that the bound $3 - \frac{4}{m+1}$ can be improved when $s_i = 1 (i = 1, \ldots, m-1)$ and $s_m = s > 1$. For $m = 2$ and 3, we will show that the worst-case performance guarantee of LS cannot be improved by any heuristic.

2. Lower bounds for on-line scheduling. For a heuristic H, let

$$R^H(m,s) = \sup_L \{C^H(L)/C^*(L) \mid L \text{ is a list of jobs and } s_i = 1 (i = 1, 2, \ldots, m-1), \quad s_m = s \geq 1 \text{ is fixed}\}, \tag{2.1}$$

$$R^H(m) = \sup_{s \geq 1} R^H(m,s). \tag{2.2}$$

In the remainder of this paper we always assume that the speed $s_i = 1 (i = 1, \ldots, m-1)$ and $s_m = s \geq 1$ if there is no special notation.

THEOREM 2.1. *The following inequalities hold.*

(i) $R^{LS}\left(2, \frac{1+\sqrt{5}}{2}\right) = R^{LS}(2) = \frac{1+\sqrt{5}}{2}$,

(ii) $R^{LS}(m,s) \leq 1 + \frac{m-1}{m+s-1} \min\{2, s\}$,

(iii) $R^{LS}(m) = R^{LS}(m,2) = 3 - \frac{4}{m+1}, \qquad (m \geq 3)$.

Proof. For the proof see [1]. ∎

THEOREM 2.2. *For any heuristic H, the following inequalities hold.*

(i) $R^H(2) \geq \frac{1+\sqrt{5}}{2}$,

(ii) $R^H(m) \geq 2, \qquad (m \geq 3)$.

Proof. Claim (i) is easily proved by using list $L_1 = \left\{1, \frac{1+\sqrt{5}}{2}\right\}$ when $s_1 = 1$ and $s_2 = \frac{1+\sqrt{5}}{2}$. Suppose that there exists an on-line algorithm H with $R^H(m) < 2$ for $m \geq 3$. Consider list $l_k = \{1, 2, \ldots, 2^k\}$ $(k = 0, 1, 2, \ldots)$. Obviously

$$R^H(m) \geq \frac{C^H(l_k)}{C^*(l_k)}$$

for any $k \geq 0$. One can easily verify that $C^*(l_k) = 2^{k-1}$. Because $R^H(m) < 2$, the jobs in list l_k must all be assigned to machine M_m; then $C^H(l_k) = 2^k - \frac{1}{2}$ and

$$R^H(m) \geq \frac{C^H(l_k)}{C^*(l_k)} = 2 - \frac{1}{2^k}.$$

We conclude that $R^H(m) \geq 2$ for the arbitrariness of k. ∎

From Theorems 2.1 and 2.2, we know that $R^{LS}(2)$ and $R^{LS}(3)$ cannot be improved anymore.

3. The algorithm. In this section we will give a heuristic. As the algorithm gets the jobs one by one, the values of C^* and C^H vary during the algorithm. To simplify notation, we will identify each job with its length. The load L_i of a machine M_i is the sum of processing times over all jobs assigned to it. In the remainder of this paper, we always assume that the speed $s_i = 1 (i = 1, 2, \ldots, m-1)$ and $s_m = s \geq 1$ if there is no special notation.

We are ready to present our heuristic A. In the algorithm, two real numbers α_m and β_m are used that satisfy $0 < \alpha_m < 1$ and $\beta_m > 1$. In order to keep the presentation simple, we drop the indices and write α and β instead. The exact values of α and β will be specified later.

ALGORITHM A.

Step 1. Reorder the machines such that $L_1 \le L_2 \le \cdots \le L_{m-1}$ holds. Let x be a new job given to the algorithm.

Step 2. Let $L = \sum_{i=1}^{m-1} L_i + sL_m$ be the total length of jobs given before x. If

$$L_m \ge \frac{(3m-5)\alpha}{s(m+1)(m-2)}L \quad \text{and} \quad x \le \frac{\beta}{m-2}L,$$

then put x on L_1.

Step 3. Assign x by LS, that is to say, assign x to the machine that will complete it first.

Throughout the following analysis, we always denote by x the current job. In order to keep the analysis simple, we present the following simple inequalities.

$$(m+s-1)C^* \ge L + x \tag{3.1}$$

and

$$t_i \le sC^* \tag{3.2}$$

for any given job t_i.

LEMMA 3.1. *If x is put on L_1 in Step* 2, *then*

$$(L_1 + x)/C^* \le \frac{m+s-1}{m-1}\left[1 + \frac{(m-2)\beta}{m+\beta-2} - \frac{(3m-5)\alpha}{(m+1)(m+\beta-2)}\right].$$

Proof. Because $x \le \frac{\beta}{m-2}L$, we have

$$x \le \frac{\beta}{m+\beta-2}(L+x). \tag{3.3}$$

Since

$$L + x = \sum_{1}^{m-1} L_i + sL_m + x$$

$$\ge (m-1)(L_1+x) + \frac{(3m-5)\alpha}{(m+1)(m-2)}(L+x) - \left[m + \frac{(3m-5)\alpha}{(m+1)(m-2)} - 2\right]x.$$

From the above and (3.3) we get

$$\left[1 + \frac{(m-2)\beta}{m+\beta-2} - \frac{(3m-5)\alpha}{(m+1)(m+\beta-2)}\right](L+x) \ge (m-1)(L_1+x). \tag{3.4}$$

By (3.1) and (3.4), Lemma 3.1 is proved. □

LEMMA 3.2. *If $x \le \frac{\beta}{m-2}L$ and x is assigned to M_i in Step* 3, *then*

$$L_i^x/C^* \le \frac{m + m\beta - 2}{m+\beta-2},$$

where L_i^x represents the load of the machine M_i after x has been assigned to M_i.

Proof. We know that in Step 3 x is assigned to either M_1 or M_m. If x is put on L_1, then $L_1 + x \le L_m + \frac{x}{s}$ and

$$\begin{aligned} L + x &= \sum_{i=1}^{m-1} L_i + sL_m + x \\ &\ge (m-1)(L_1 + x) + s\left(L_m + \frac{x}{s}\right) - (m-1)x \\ &\ge (m+s-1)(L_1 + x) - \frac{(m-1)\beta}{m+\beta-2}(L+x). \end{aligned}$$

From the above and (3.1), we get

$$(L_1 + x)/C^* \le \frac{m + m\beta - 2}{m + \beta - 2}.$$

Similarly we can prove

$$\left(L_m + \frac{x}{s}\right)/C^* \le \frac{m + m\beta - 2}{m + \beta - 2}$$

if x is put on L_m. □

In the remainder of this paper, the real number $\bar{\alpha}$ satisfies that $\bar{\alpha} > \alpha$.

LEMMA 3.3. *If*

$$\frac{\beta}{m-2}L < x \le \frac{\bar{\beta}}{m-1}L \quad \textit{and} \quad L_m < \frac{(3m-5)\bar{\alpha}}{s(m+1)(m-2)}L$$

then

$$L_i^x/C^* \le \frac{(m+s-1)(m-1)}{(m+\bar{\beta}-1)s}\left[\frac{(3m-5)\bar{\alpha}}{(m+1)(m-2)} + \frac{\bar{\beta}}{(m-1)}\right].$$

Proof. From (3.1) we have

$$\begin{aligned} \left(L_m + \frac{x}{s}\right)\Big/ C^* &\le \left(L_m + \frac{x}{s}\right)\Big/ \frac{L+x}{m+s-1} \\ &\le (m+s-1)\left(L_m + \frac{\bar{\beta}L}{s(m-1)}\right)\Big/\left(L + \frac{\bar{\beta}}{m-1}L\right) \\ &\le \frac{(m+s-1)(m-1)}{(m+\bar{\beta}-1)s}\left[\frac{(3m-5)\bar{\alpha}}{(m+1)(m-2)} + \frac{\bar{\beta}}{(m-1)}\right]. \end{aligned}$$

Because x is assigned in Step 3, $L_i^x \le L_m + \frac{x}{s}$. Lemma 3.3 is proved. □

LEMMA 3.4. *If x is assigned to machine M_i and*

$$L_m \ge \frac{(3m-5)\bar{\alpha}}{s(m+1)(m-2)}L, \qquad \frac{\beta}{m-2}L \le x < \frac{\bar{\beta}}{m-1}L$$

and

$$\frac{(3m-5)\alpha}{(m+1)(m+\beta-2)} + \frac{\beta}{m+\beta-2} \le \frac{(3m-5)\bar{\alpha}}{(m+1)(m-2)}, \tag{3.5}$$

then

$$L_i^x/C^* \le \max\left\{\frac{s(m-2)(m\bar{\beta}+m-1)(m\beta+m-2)(m+1)}{(m-1)(m+s-1)^2(3m-5)\bar{\alpha}\beta},\right.$$
$$\left.1+\frac{m-1}{m+s-1}\max\{1,s-1\}\right\}.$$

Proof. Let t be the last task assigned to machine M_m before x.

Case 1. $t \le C^*$. Let $\bar{L}_i (i=1,2,\ldots,m)$ be the machine $M_i's$ load before t is assigned and let $\bar{L}$ be the total length of tasks before t is given. If

$$t \le \frac{\beta}{m-2}\bar{L}, \quad \text{then } \bar{L}_m < \frac{(3m-5)\alpha}{s(m+1)(m-2)}\bar{L}.$$

So

$$\begin{aligned} L_m &= \bar{L}_m + \frac{t}{s} < \frac{(3m-5)\alpha}{s(m+1)(m-2)}\bar{L} + \frac{t}{s} \\ &= \frac{(3m-5)\alpha}{s(m+1)(m-2)}(\bar{L}+t) + \frac{1}{s}\left[1-\frac{(3m-5)\alpha}{(m+1)(m-2)}\right]t \\ &\le \frac{(3m-5)\alpha}{s(m+1)(m-2)}(\bar{L}+t) + \frac{1}{s}\left(1-\frac{(3m-5)\alpha}{(m+1)(m-2)}\right)\frac{\beta}{m+\beta-2}(\bar{L}+t) \\ &\le \frac{1}{s}\left[\frac{(3m-5)\alpha}{(m+1)(m+\beta-2)}+\frac{\beta}{m+\beta-2}\right]L. \end{aligned}$$

This is a contradiction because

$$L_m \ge \frac{(3m-5)\bar{\alpha}}{s(m+1)(m-2)}L \ge \frac{1}{s}\left[\frac{(3m-5)\alpha}{(m+1)(m+\beta-2)}+\frac{\beta}{m+\beta-2}\right]L.$$

So we get

$$t > \frac{\beta}{m-2}\bar{L} \tag{3.6}$$

and t is assigned as in Step 3. Since

$$\begin{aligned} \bar{L}+t &\ge (m-1)\bar{L}_1 + s\bar{L}_m + t \\ &= (m-1)(\bar{L}_1+t) + s\left(\bar{L}_m+\frac{t}{s}\right) - (m-1)t \\ &\ge (m+s-1)\left(\bar{L}_m+\frac{t}{s}\right) - (m-1)t \\ &= (m+s-1)L_m - (m-1)t, \end{aligned}$$

and by

$$L_m \ge \frac{(3m-5)\bar{\alpha}}{s(m+1)(m-2)}L$$

and (3.6), we conclude that

$$t \geq \frac{\beta\bar{\alpha}(3m-5)(m+s-1)}{s(\beta m+m-2)(m+1)(m-2)}L. \tag{3.7}$$

If x is assigned to M_m, then

$$\begin{aligned} L+x &= \sum_{i=1}^{m-1} L_i + sL_m + x \\ &\geq (m-1)(L_1+x) + s\left(L_m+\frac{x}{s}\right) - (m-1)x \\ &\geq (m+s-1)\left(L_m+\frac{x}{s}\right) - \frac{(m-1)\bar{\beta}}{m+\bar{\beta}-1}(L+x). \end{aligned}$$

From the above we get

$$L+x \geq \frac{(m+s-1)(m+\bar{\beta}-1)}{m\bar{\beta}+m-1}\left(L_m+\frac{x}{s}\right).$$

So

$$\begin{aligned} \left(L_m+\frac{x}{s}\right)/C^* &\leq \left(L_m+\frac{x}{s}\right)\Big/ t \\ &\leq \frac{m\bar{\beta}+m-1}{(m+s-1)(m+\bar{\beta}-1)}\frac{L+x}{t} \\ &\leq \frac{m\bar{\beta}+m-1}{(m+s-1)(m+\bar{\beta}-1)}\cdot\left(L+\frac{\bar{\beta}}{m-1}L\right) \\ &\quad\cdot\frac{s(m\beta+m-2)(m+1)(m-2)}{\beta\bar{\alpha}(3m-5)(m+s-1)L} \\ &= \frac{s(m\bar{\beta}+m-1)(m\beta+m-2)(m+1)(m-2)}{\beta\bar{\alpha}(m+s-1)^2(m-1)(3m-5)}. \end{aligned}$$

Case 2. $t > C^*$. In this case, we conclude that

$$x \leq C^*\max\{1, s-1\}.$$

So

$$\begin{aligned} (m+s-1)C^* \geq L+x &\geq (m-1)L_1 + s\left(L_m+\frac{x}{s}\right) \\ &\geq (m+s-1)\left(L_m+\frac{x}{s}\right) - (m-1)x \\ &\geq (m+s-1)\left(L_m+\frac{x}{s}\right) - (m-1)C^*\max\{1, s-1\}. \end{aligned}$$

From the above we get

$$\left(L_m+\frac{x}{s}\right)\Big/ C^* \leq 1 + \frac{m-1}{m+s-1}\max\{1, s-1\}.$$

Similarly, we can prove it if x is assigned to M_1. □

LEMMA 3.5. *If* $x > \frac{\bar{\beta}}{m-1}L$ *and* x *is assigned to* M_i *in Step* 3, *then*

$$L_i^x/C^* \le \frac{sm}{m+s+1} + \frac{s(m-1)}{\bar{\beta}(m+s-1)},$$

where L_i^x *represents the load of machine* M_i *after* x *is assigned to* M_i.

Proof. If x is assigned to M_1, then

$$\begin{aligned} L + x &= \sum_{i=1}^{m-1} L_i + sL_m + x \\ &\ge (m+s-1)(L_1 + x) - (m-1)x. \end{aligned}$$

From the above, $x \le sC^*$, and $x > \frac{\bar{\beta}}{m-1}L$ we have

$$\begin{aligned} \frac{L_1 + x}{C^*} &\le (L_1 + x) \Big/ \frac{x}{s} \le \frac{s(L+mx)}{(m+s-1)x} \\ &\le \frac{ms}{m+s-1} + \frac{sL}{m+s-1} \cdot \frac{m-1}{\bar{\beta}L} \\ &= \frac{ms}{m+s-1} + \frac{(m-1)s}{(m+s-1)\bar{\beta}}. \end{aligned}$$

If x is assigned to M_m, the proof is similar. □

LEMMA 3.6. *Let* $R_m = \min R$, *subject to*

$$\begin{cases} (m+1)R^2 - 2(m-1)R - 2m > 0 \\ 2 < R < \dfrac{3m-1}{m+1} \\ \dfrac{(m+1)R^2 - 2(m-1)R - m - 1}{(m+1)^2 R - 2m(m+1)} \\ \qquad \ge \dfrac{1}{2}\left[\dfrac{(m+1)R^2 - 2(m-1)R - m - 1}{(m+1)R^2 - 2(m-1)R - 2m} - \dfrac{(m-1)R}{m+1}\right]. \end{cases} \tag{3.8}$$

We have that R_m *is the only real root of*

$$\begin{aligned} (m+1)^3 R^4 - (m+1)(5m^2 + 4m - 5)R^3 + (6m^3 - 18m + 4)R^2 \\ + (m^3 + 11m^2 - m - 3)R - 2m(m^2 - 1) = 0 \end{aligned} \tag{3.9}$$

in interval $(2, 3 - \frac{4}{m+1})$, *and* R_m *is an increasing function of* m. $\lim_{m\to\infty} R_m$ *is the only real root of equation* $R^3 - 3R^2 + 1 = 0$ *between* 2 *and* 3.

Proof. Equation (3.8) is equivalent to

$$\begin{aligned} \frac{1}{2}R + \frac{2}{(m+1)^2} - \frac{(m-1)^2}{(m+1)^2[(m+1)R - 2m]} \\ \ge \frac{m-1}{2[(m+1)R^2 - 2(m-1)R - 2m]} + \frac{1}{2}. \end{aligned} \tag{3.10}$$

If m is fixed, the left-hand side of (3.10) is an increasing function of R and the right-hand side of (3.10) is a decreasing function of R in interval

$$\left(2, \frac{m-1+\sqrt{3m^2+1}}{m+1}\right)$$

and

$$\left(\frac{m-1+\sqrt{3m^2+1}}{m+1}, \frac{3m-1}{m+1}\right],$$

respectively. If $R = 2$ or $\frac{3m-1}{m+1}$, the inequality (3.10) is strict and

$$\frac{m-1+\sqrt{3m^2+1}}{m+1}$$

is an odd point of the right-hand side. So R_m exists and is the only real root of equation

$$\frac{1}{2}R + \frac{2}{(m+1)^2} - \frac{(m-1)^2}{(m+1)^2((m+1)R-2m)} = \frac{m-1}{2[(m+1)R^2-2(m-1)R-2m]} + \frac{1}{2}. \tag{3.11}$$

If R is fixed, it is easy to verify that the right-hand side of (3.10) is an increasing function of m but the left-hand side is a decreasing function of m. So we can conclude that R_m is an increasing function of m. From (3.11) we get (3.9), and letting m tend to infinity in (3.11), we get $R^3 - 3R^2 + 1 = 0$. □

THEOREM 3.7. *For Algorithm A there exists $\varepsilon_m > 0$ such that*

$$R^A(m,2) \le \frac{3m-1}{m+1} - \varepsilon_m$$

and there exists a positive ε such that $\varepsilon_m \ge \varepsilon$ for every $m \ge 4$.

Proof. Lemmas 3.1–3.5 give upper bounds on the worst-case ratios in the five scenarios. If $\alpha < 1$ and $\beta > 1$, we can easily verify that

$$\frac{m+1}{m-1}\left[1 + \frac{(m-2)\beta}{m+\beta-2} - \frac{(3m-5)\alpha}{(m+1)(m+\beta-2)}\right] > \frac{m+m\beta-2}{m+\beta-2}. \tag{3.12}$$

Let R be the real root of equation (3.9), and

$$\alpha_1 = \frac{2(m+1)^2R^4 - 8(m^2-1)R^3 + 2(3m^2-10m+3)R^2 + (3m^2-2m-5)R + 2m(m+1)}{[(m+1)R-2m][(m+1)R^2-2(m-1)R-2m-1]},$$

$$\alpha = \frac{m-2}{3m-5}\alpha_1,$$

$$\beta = \frac{m-2}{(m+1)R^2-2(m-1)R-2m-1},$$

$$\bar{\alpha} = \frac{2(m-2)[(m+1)R^2-2(m-1)R-m-1]}{(3m-5)[(m+1)R-2m]},$$

$$\bar{\beta} = \frac{2(m-1)}{(m+1)R-2m};$$

TABLE 3.1

m	α_m	β_m	$R^{LS}(m,2)$	$R^{A}(m,2)$
4	0.9482	1.1512	2.2	2.1835
5	0.8951	1.2557	2.3333	2.3025
9	0.7634	1.4854	2.6	2.5353
∞	0.5357	1.8779	3	2.8795

then inequality (3.5) becomes equality and the upper bounds given in Lemma 3.1, 3.3, 3.4, and 3.5 are all equal to R. From Lemma 3.6, let $\varepsilon_m = \frac{3m-1}{m+1} - R$; then we get

$$R^A(m,2) \le R = \frac{3m-1}{m+1} - \varepsilon_m.$$

Because ε_m tends to a positive number when m tends to infinity, there exists a positive number ε such that $\varepsilon_m \ge \varepsilon$ for every $m \ge 4$. □

The comparison for some m between the two algorithms is shown in Table 3.1.

In the following we devise an algorithm $\bar{A}$ for general $s_m = s > 1$.

ALGORITHM $\bar{A}$. Let ε_1 and ε_2 be two positive numbers. If $2 - \varepsilon_1 \le s \le 2 + \varepsilon_2$, then we use algorithm A to schedule; otherwise we use algorithm LS.

THEOREM 3.8. *There exist suitable* $\varepsilon_1 > 0$, $\varepsilon_2 > 0$, *and* $\varepsilon_m > 0$ *such that*

$$R^{\bar{A}}(m) \le \frac{3m-1}{m+1} - \varepsilon_m$$

for every $m \ge 4$.

Proof. Because the upper bounds given in Lemmas 3.1, 3.2, 3.4, and 3.5 are continuous functions of s, from Theorems 2.1 and 3.7, Theorem 3.8 is proved. □

4. Conclusion. In this paper we derived two on-line algorithms that beat list scheduling in the measure of worst-case performance (for $m \ge 4$) on two conditions, respectively. However, the following question may be very interesting. First, asymptotically our analysis did not improve the heuristic LS in Theorem 3.8, since ε_m may tend to zero as m tends to infinity. Second, in Theorem 3.8 $R^{\bar{A}}(m,s) = R^{LS}(m,s)$ for most $s > 1$. Are there on-line scheduling algorithms with worst case better than $R^{LS}(m,s)$ for any fixed $s \ge 1$? Third, for $m \ge 4$ machines, we gave a lower bound of 2. How can we get a greater lower bound?

Acknowledgment. We are grateful to our supervisor, Professor Minyi Yue, for his encouragement on this problem.

REFERENCES

[1] Y. CHO AND S. SAHNI, *Bounds for list schedules on uniform processors*, SIAM J. Comput., 9 (1980), pp. 91–103.

[2] R. L. GRAHAM, *Bounds on multiprocessing timing anomalies*, SIAM J. Appl. Math., 17 (1969), pp. 416–429.

[3] G. GALAMBOS AND G. J. WOEGINGER, *An on-line scheduling heuristic with better worst case ratio than Graham's List Scheduling*, SIAM J. Comput., 22 (1993), pp. 349–355.

[4] Y. BARTAL, A. FIAT, H. KARLOFF, AND R. VOHRA, *New algorithms for an ancient scheduling problem*, in Proc. 24th ACM Symp. on Theory of Computing, ACM, New York, 1992, pp. 51–58.

SIAM J. COMPUT.
Vol. 27, No. 2, pp. 423–447, April 1998
© 1998 Society for Industrial and Applied Mathematics
006

THE LOAD, CAPACITY, AND AVAILABILITY OF QUORUM SYSTEMS*

MONI NAOR† AND AVISHAI WOOL‡

Abstract. A *quorum system* is a collection of sets (quorums) every two of which intersect. Quorum systems have been used for many applications in the area of distributed systems, including mutual exclusion, data replication, and dissemination of information.

Given a strategy to pick quorums, the load $\mathcal{L}(\mathcal{S})$ is the minimal access probability of the busiest element, minimizing over the strategies. The capacity *Cap*(S) is the highest quorum accesses rate that $\mathcal{S}$ can handle, so $Cap(\mathcal{S}) = 1/\mathcal{L}(\mathcal{S})$.

The availability of a quorum system $\mathcal{S}$ is the probability that at least one quorum survives, assuming that each element fails independently with probability p. A tradeoff between $\mathcal{L}(\mathcal{S})$ and the availability of $\mathcal{S}$ is shown.

We present four novel constructions of quorum systems, all featuring optimal or near optimal load, and high availability. The best construction, based on paths in a grid, has a load of $O(1/\sqrt{n})$, and a failure probability of $\exp(-\Omega(\sqrt{n}))$ when the elements fail with probability $p < \frac{1}{2}$. Moreover, even in the presence of faults, with exponentially high probability the load of this system is still $O(1/\sqrt{n})$. The analysis of this scheme is based on percolation theory.

Key words. quorum systems, load, fault tolerance, distributed computing, percolation theory, linear programming

AMS subject classifications. 60K35, 62N05, 68M10, 68Q22, 68R05, 90A28, 90C05

PII. S0097539795281232

1. Introduction.

1.1. Motivation. *Quorum systems* serve as basic tools providing a uniform and reliable way to achieve coordination between processors in a distributed system. Quorum systems are defined as follows. A *set system* is a collection of sets $\mathcal{S} = \{S_1, \ldots, S_m\}$ over an underlying universe $U = \{u_1, \ldots, u_n\}$. A set system is said to satisfy the *intersection property* if every two sets $S, R \in \mathcal{S}$ have a nonempty intersection. Set systems with the intersection property are known as *quorum systems*, and the sets in such a system are called quorums.

Quorum systems have been used in the study of problems such as *mutual exclusion* (cf. [39]), *data replication protocols* (cf. [7, 18]), *name servers* (cf. [32]), *selective dissemination of information* (cf. [46]), and *distributed access control and signatures* (cf. [34]).

A protocol template based on quorum systems works as follows. In order to perform some action (e.g., update the database, enter a critical section), the user selects a quorum and *accesses all its elements.* The intersection property then guarantees that the user will have a consistent view of the current state of the system. For example,

*Received by the editors February 8, 1995; accepted for publication (in revised form) January 30, 1996. A preliminary version of this paper appeared in the 35th Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 214–225.

http://www.siam.org/journals/sicomp/27-2/28123.html

†Department of Applied Mathematics and Computer Science, The Weizmann Institute, Rehovot 76100, Israel (naor@wisdom.weizmann.ac.il). This author is an incumbent of the Morris and Rose Goldman Career Development Chair; research was supported by an Alon fellowship and by a grant from the Israel Science Foundation administered by the Israeli Academy of Sciences and Humanities.

‡Bell Laboratories, Lucent Technologies, 700 Mountain Avenue, Murray Hill, NJ 07974 (yash@research.bell-labs.com). The research of this author was completed at the Department of Applied Mathematics and Computer Science, The Weizmann Institute, Rehovot 76100, Israel.

if all the members of a certain quorum give the user permission to enter the critical section, then any other user trying to enter the critical section before the first user has exited (and released the permission-granting quorum from its lock) will be refused permission by at least one member of any quorum it chooses to access.

In this work we consider three criteria of measuring how good a quorum system is:

1. `Load`. A strategy is a rule giving each quorum an access frequency (so that the frequencies sum up to 1). A strategy induces a load on each element, which is the sum of the frequencies of all quorums it belongs to. This represents the fraction of the time an element is used. For a given quorum system $\mathcal{S}$, the load $\mathcal{L}(\mathcal{S})$ is the minimal load on the busiest element, minimizing over the strategies. The load measures the quality of a quorum system in the following sense. If the load is low, then each element is accessed rarely; thus it is free to perform other unrelated tasks.

2. `Capacity`. We would like the system to handle as many requests as possible. For this purpose we define $a(\mathcal{S}, k)$ as the number of quorum accesses that $\mathcal{S}$ can handle during a period of k time units. This is the maximal t for which there exists a way to schedule t quorum accesses, to quorums $S^1, \ldots, S^t$ (allowing repetitions), such that no element is accessed more than k times. The capacity $Cap(\mathcal{S})$ is then the limit as $k \to \infty$ of $a(\mathcal{S}, k)$ normalized by k.

3. `Availability`. Assuming that each element fails with probability p, what is the probability, F_p, that the surviving elements do not contain any quorum? This failure probability measures how resilient the system is, and we would like F_p to be as small as possible.

Our goal is to investigate these criteria and find quorum systems that perform well according to all three.

1.2. Related work. The first distributed control protocols using quorum systems [42, 14] use *voting* to define the quorums. Each processor has a number of votes, and a quorum is any set of processors with a combined number of votes exceeding half of the system's total number of votes. The simple majority system is the most obvious voting system.

The availability of voting systems is studied in [4]. It is shown that in terms of availability, the majority is the best quorum system when $p < \frac{1}{2}$. In [35] the failure probability function F_p is characterized, and among other things it is shown that the singleton has the best availability when $p > \frac{1}{2}$. The case when the elements fail with different probabilities p_i, all less than $\frac{1}{2}$, is addressed in [41].

The first paper to explicitly consider mutual exclusion protocols in the context of intersecting set systems was [13]. In this work the term *coterie* and the concept of *domination* are introduced. Several basic properties of dominated and nondominated coteries are proved.

Alternative protocols based on quorum systems (rather than on voting) appear in [28] (using finite projective planes), [1] (the Tree system), [5, 25] (using a grid), and [23, 24, 38] (hierarchical systems). The triangular system is due to [26, 9]. The Wheel system appears in [29]. The CWlog system appears in [37, 36]. The motivation for several of these alternative systems was to find constructions with high availability and low load (which is referred to in most of these papers as quorum systems with small quorums).

In [19], the question of how evenly balanced the work load can be is studied. Tradeoffs between the potential load balancing of a system and its average load are obtained, and it is shown that in some quorum systems it is impossible to have a perfect load balance in which all the elements have an equal load.

A concept of capacity in voting systems is defined in [21] and some voting systems are compared. The analysis does not distinguish between properties of the quorum system and properties of the strategy that chooses which quorum to access.

A good reference to percolation theory is [15]. Two successful applications of percolation to questions of computer science are shown in [30] and [8].

While the majority quorum system is the best in terms of availability, and the finite projective planes construction have excellent load, they fail miserably according to the other criteria: the load of majority is $\frac{1}{2}$ and the failure probability of the projective planes (FPP) goes to 1 (quickly) as the number of elements grows. In fact, all of the existing constructions are less successful than ours in the simultaneous achievement of high availability and low load.

1.3. New results. We start by defining the concepts of load and capacity and showing that they can be formulated as linear or integer linear programs. Then using results of hypergraph theory we show that $Cap(\mathcal{S}) = 1/\mathcal{L}(\mathcal{S})$. Therefore, all the information regarding the capacity is captured by $\mathcal{L}(\mathcal{S})$.

We obtain several lower bounds on the load $\mathcal{L}(\mathcal{S})$. We show that if the minimal quorum size is $c(\mathcal{S})$, then $\mathcal{L}(\mathcal{S}) \geq \max\{1/c(\mathcal{S}), c(\mathcal{S})/n\}$; hence $\mathcal{L}(\mathcal{S}) \geq 1/\sqrt{n}$. We also obtain a tradeoff between the load and failure probability, i.e., $F_p \geq p^{n\mathcal{L}(\mathcal{S})}$. In some cases the linear program formulation of load also allows us to efficiently compute the load of a given quorum system, even if the quorums are not represented explicitly, using the ellipsoid algorithm adaptation of [16]. The behavior of the load when the elements may fail is also studied. We assume the common model that the elements fail independently with probability p. The load then becomes a random variable $\mathcal{L}_p(\mathcal{S})$.

Next we show some conditions that prove that a given strategy w induces the optimal load. This enables us to find optimal strategies and to calculate $\mathcal{L}(\mathcal{S})$ of some quorum systems without actually solving the linear program.

The major contributions of this work are four novel quorum system constructions, all of which have optimal or near optimal load *and* high availability, i.e., a failure probability that tends to 0 exponentially fast when $p < \frac{1}{2}$, or at least when $p < \beta < \frac{1}{2}$. Our best construction is the Paths system, which is based on a percolation grid. It has a load of $O(1/\sqrt{n})$, and a failure probability of $\exp(-\Omega(\sqrt{n}))$ when the elements fail with probability $p < \frac{1}{2}$. Moreover, even in the presence of faults, with exponentially high probability the load of this system is still $O(1/\sqrt{n})$. Two other constructions resemble the Grid construction but are enhanced so their failure probability tends to 0. The B-Grid system has $\mathcal{L}(\text{B-Grid}) = O(1/\sqrt{n})$ and if $p < \frac{1}{3}$, then $F_p(\text{B-Grid}) = O(\exp(-n^{1/4}/2))$. The SC-Grid system has $\mathcal{L}(\text{SC-Grid}) = O(\sqrt{(\ln n)/n})$, and if $p < \frac{1}{2} - \delta$ for some $\delta > 0$, then $F_p(\text{SC-Grid}) \leq \exp(-\Omega(\sqrt{n \ln n}))$. The AndOr system uses the AND/OR trees of [43]. It has $\mathcal{L}(\text{AndOr}) = O(1/\sqrt{n})$, $F_p(\text{AndOr}) \leq \exp(-\Omega(\sqrt{n}))$ when $p < \frac{1}{4}$, and $F_p \leq \exp(-\Omega(n^{0.19}))$ if $p \leq 0.38 - \Omega(n^{-0.19})$. The three latter constructions also enjoy the property that their quorums are all of size $O(\sqrt{n})$.

Finally, we analyze the load of some known quorum system constructions. We show that all voting systems have a load of at least $\frac{1}{2}$, which is very high. We also show that nondominated coteries have lower load than dominated ones.

The paper is organized in as follows. In section 2 we present some basic definitions. In section 3 we define the load and the capacity, their linear programs, and the relationship between them. In section 4 we prove the basic properties of the load. In section 5 we present the new constructions. In section 6 we analyze the load of some quorum systems.

2. Preliminaries.

2.1. Definitions and notation.

DEFINITION 2.1. *A* set system $\mathcal{S} = \{S_1, \ldots, S_m\}$ *is a collection of subsets* $S_i \subseteq U$ *of a finite universe* U. *A* quorum system *is a set system* $\mathcal{S}$ *that has the* intersection property: $S \cap R \neq \varnothing$ *for all* $S, R \in \mathcal{S}$.

Alternatively, quorum systems are known as *intersecting set systems* or *intersecting hypergraphs*. The sets of the system are called *quorums*. The number of elements in the underlying universe is denoted by $n = |U|$. The number of quorums in the system is denoted by m. The cardinality of the smallest quorum in $\mathcal{S}$ is denoted by $c(\mathcal{S}) = \min\{|S| : S \in \mathcal{S}\}$.

The *degree* of an element $i \in U$ in a quorum system $\mathcal{S}$ is the number of quorums that contain i: $\deg(i) = |\{S \in \mathcal{S} : i \in S\}|$.

DEFINITION 2.2. *Let* $\mathcal{S}$ *be a quorum system.* $\mathcal{S}$ *is* s-uniform *if* $|S| = s$ *for all* $S \in \mathcal{S}$.

DEFINITION 2.3. *A quorum system* $\mathcal{S}$ *is* (s,d)-fair *if it is* s*-uniform and* $\deg(i) = d$ *for all* $i \in U$. $\mathcal{S}$ *is called* s*-fair if it is* (s,d)*-fair for some* d.

DEFINITION 2.4. *A* coterie *is a quorum system* $\mathcal{S}$ *that has the* minimality property: *there are no* $S, R \in \mathcal{S}$, $S \subset R$.

DEFINITION 2.5. *Let* $\mathcal{R}, \mathcal{S}$ *be coteries* (*over the same universe* U). *Then* $\mathcal{R}$ dominates $\mathcal{S}$, *denoted* $\mathcal{R} \succ \mathcal{S}$, *if* $\mathcal{R} \neq \mathcal{S}$ *and for each* $S \in \mathcal{S}$ *there is* $R \in \mathcal{R}$ *such that* $R \subseteq S$. *A coterie* $\mathcal{S}$ *is called* dominated *if there exists a coterie* $\mathcal{R}$ *such that* $\mathcal{R} \succ \mathcal{S}$. *If no such coterie exists, then* $\mathcal{S}$ *is* nondominated (ND). *Let* NDC *denote the class of all* ND *coteries.*

2.2. The probabilistic failure model. We use a simple probabilistic model of the failures in the system. We assume that the elements (processors) fail independently with probabilities p_i. We assume that the failures are *transient*. We assume also that the failures are *crash* failures and that they are *detectable*.

DEFINITION 2.6. *A* configuration *is a vector* $\mathbf{x} \in \{0,1\}^n$ *in which* $x_i = 1$ *iff the element* $i \in U$ *has* failed.

Notation. For a configuration $\mathbf{x}$ let $dead(\mathbf{x}) = \{i \in U : x_i = 1\}$ denote the set of failed elements, and let $live(\mathbf{x}) = \{i \in U : x_i = 0\}$ denote the set of functioning elements.

Notation. We use $q_i = 1 - p_i$ to denote the probability of survival of element i.

DEFINITION 2.7. *For every quorum* $S \in \mathcal{S}$ *let* $\mathcal{E}_S$ *be the event that* S *is* hit, *i.e., at least one element* $i \in S$ *has failed* (*or,* $x_i = 1$ *for some* $i \in S$). *Let* $fail(\mathcal{S})$ *be the event that all the quorums* $S \in \mathcal{S}$ *are hit, i.e.,* $fail(\mathcal{S}) = \bigcap_{S \in \mathcal{S}} \mathcal{E}_S$.

When the failure probabilities are equal, i.e., $\mathbf{p} = (p, \ldots, p)$, we use the definition of [35] of the global system failure probability of a quorum system $\mathcal{S}$, as follows.

DEFINITION 2.8. $F_p(\mathcal{S}) = \mathbb{P}_p(fail(\mathcal{S})) = \mathbb{P}_p(\bigcap_{S \in \mathcal{S}} \mathcal{E}_S)$.

When we consider the asymptotic behavior of $F_p(\mathcal{S}_n)$ for a sequence $\mathcal{S}_n$ of quorum system over a universe with an increasing size n, we find that for many constructions it is similar to the behavior described by the Condorcet Jury Theorem [6]. Hence, we have the following definition of [35].

DEFINITION 2.9. *A parameterized family of functions* $g_p(n) : \mathbb{N} \to [0,1]$, *for* $p \in [0,1]$, *is said to be* Condorcet *iff*

$$\lim_{n \to \infty} g_p(n) = \begin{cases} 0, & p < \frac{1}{2}, \\ 1, & p > \frac{1}{2}, \end{cases}$$

and $g_{1/2}(n) = \frac{1}{2}$ *for all* n. *If* $g_p(n)$ *has this behavior for* $p \neq \frac{1}{2}$ *but* $g_{1/2}(n) \neq \frac{1}{2}$, *then it is said to be* almost Condorcet.

3. Load and capacity.

3.1. Strategies and load. A protocol using a quorum system (for mutual exclusion, say) occasionally needs to access quorums during its run. A strategy is a probabilistic rule that governs which quorum is chosen each time. In other words, a strategy gives the frequency of picking each quorum S_j.

DEFINITION 3.1. *Let a quorum system* $\mathcal{S} = (S_1, \ldots, S_m)$ *be given over a universe* U. *Then* $w \in [0,1]^m$ *is a* strategy *for* $\mathcal{S}$ *if it is a probability distribution over the quorums* $S_j \in \mathcal{S}$, *i.e.,* $\sum_{j=1}^m w_j = 1$.

For every element $i \in U$, a strategy w of picking quorums induces the frequency of accessing element i, which we call the *load* on i. The *system load*, $\mathcal{L}(\mathcal{S})$, is the load on the *busiest* element induced by the *best* possible strategy. Formally, we have the following definition.

DEFINITION 3.2. *Let a strategy* w *be given for a quorum system* $\mathcal{S} = (S_1, \ldots, S_m)$ *over a universe* U. *For an element* $i \in U$, *the* load *induced by* w *on* i *is* $\ell_w(i) = \sum_{S_j \ni i} w_j$. *The* load *induced by a strategy* w *on a quorum system* $\mathcal{S}$ *is*

$$\mathcal{L}_w(\mathcal{S}) = \max_{i \in U} \ell_w(i).$$

The system load *on a quorum system* $\mathcal{S}$ *is*

$$\mathcal{L}(\mathcal{S}) = \min_w \{\mathcal{L}_w(\mathcal{S})\},$$

where the minimum is taken over all strategies w.

Remarks.

(i) The load $\mathcal{L}(\mathcal{S})$ should be viewed as a "best case" definition. A load of $\mathcal{L}(\mathcal{S})$ is achieved only if the quorums are chosen according to an optimal strategy. However, a protocol using the quorum system may use some other strategy (for instance, if computing an optimal strategy is too costly) hence, the actual load might be higher than $\mathcal{L}(\mathcal{S})$. This also means that $\mathcal{L}(\mathcal{S})$ is a property inherent in the combinatorial structure of the *quorum system* and not in the protocol using the system.

(ii) In the definition of $\mathcal{L}(\mathcal{S})$ we are assuming that all the elements of the universe are functioning, so all the quorums of the system are usable. In the following the definition is extended to the case where the elements may fail.

3.2. A linear programming formulation of the load. An alternative way to define the load is via a linear programming formulation. This formulation shows that the load $\mathcal{L}(\mathcal{S})$ can be computed in polynomial time using linear programming algorithms (cf. [40]) if $\mathcal{S}$ is given explicitly.

DEFINITION 3.3. *Let a quorum system* $\mathcal{S} = (S_1, \ldots, S_m)$ *be given over a universe* U *of size* n. *Define a variable* w_j *for each quorum* $S_j \in \mathcal{S}$ *and an additional variable* L. *Then the* system load $\mathcal{L}(\mathcal{S})$ *is defined by the following linear program.*

$$LP : \mathcal{L}(\mathcal{S}) = \min L, \quad s.t. \begin{cases} \sum_{j=1}^m w_j = 1, & \text{(i)} \\ \sum_{S_j \ni i} w_j \leq L, \quad \textit{for all } i \in U, & \text{(ii)} \\ w_j \geq 0,\ L \geq 0. & \text{(iii)} \end{cases}$$

Notation. We use $(w; L)$ to denote a tuple of a strategy and a possible load that together constitute a point in the problem domain $[0,1]^{m+1}$.

Remark. The program LP is always feasible, since for any quorum system $\mathcal{S}$ and strategy w, the point $(w; 1)$ is trivially feasible. Clearly, LP is also a bounded linear program, so $\mathcal{L}(\mathcal{S})$ is always finite.

The next definition and lemma show that the load $\mathcal{L}(\mathcal{S})$ is closely related to the well-known *fractional matching number* of a hypergraph (cf. [12, p. 149]).

DEFINITION 3.4. *The fractional matching number of a quorum system, denoted by ν^*, is*

$$FM : \nu^*(\mathcal{S}) = \max \sum_{j=1}^{m} f_j, \quad s.t. \begin{cases} \sum_{S_j \ni i} f_j \leq 1, & \text{for all } i \in U, \\ f_j \geq 0. \end{cases}$$

LEMMA 3.5. *$\mathcal{L}(\mathcal{S}) = 1/\nu^*(\mathcal{S})$ for any quorum system $\mathcal{S}$.*

Proof. Let w be an optimal strategy for program LP, attaining $\mathcal{L}(\mathcal{S})$. Then f defined by $f_j = w_j/\mathcal{L}(\mathcal{S})$ is feasible in program FM. Since FM is maximizing, it follows that $\nu^*(\mathcal{S}) \geq \sum_j f_j = 1/\mathcal{L}(\mathcal{S})$.

On the other hand, consider f which optimizes program FM, with $\sum_j f_j = \nu^*(\mathcal{S})$. Then w defined by $w_j = f_j/\nu^*$ is a strategy (since $\sum_j w_j = 1$), and $(w; 1/\nu^*)$ is feasible for program LP. Since $\mathcal{L}(\mathcal{S})$ is minimal it follows that $\mathcal{L}(\mathcal{S}) \leq 1/\nu^*(\mathcal{S})$. □

Notation. For a vector $\mathbf{y} \in [0, 1]^n$ and a set $S \subseteq U$, let $\mathbf{y}(S) = \sum_{i \in S} y_i$.

FACT 3.6. *Let $\mathcal{S}$ be a quorum system as in Definition 3.3. Define a variable y_i for each element $i \in U$, and an additional variable T. The dual of program LP is*

$$DLP : t(\mathcal{S}) = \max T, \quad s.t. \begin{cases} \sum_{i=1}^{n} y_i \leq 1, & \text{(iv)} \\ \mathbf{y}(S_j) \geq T, \quad \text{for all } S_j \in \mathcal{S}, & \text{(v)} \\ y_i \geq 0, & \text{(vi)} \\ T \lneqq 0. & \text{(vii)} \end{cases}$$

Remarks.

(i) Formally the variable T is unconstrained (vii). However, at the optimum, $t(\mathcal{S}) = T$ is positive, since $T = 0$ is feasible for any vector $\mathbf{y} \in [0, 1]^n$ and DLP is a maximization problem.

(ii) The value of $t(\mathcal{S})$ does not change if we require equality in (iv), since we can increase the y_i values without violating any inequality in (v) and without changing T.

Using the dual program DLP allows us in some cases to compute $\mathcal{L}(\mathcal{S})$, even when $\mathcal{S}$ is given implicitly, using the ellipsoid algorithm of [16, 27] (see section 4.3).

3.3. The capacity of a quorum system. Each time that a distributed protocol generates an access to a quorum $S \in \mathcal{S}$, it causes work to be done by the elements of S. During the time that the elements of S are busy with one quorum access, they cannot handle another. However, other elements may be used in the next quorum access, making use of the parallelism in the system. We want to find what is the maximal rate of quorum access that the system allows.

Assume that it takes one unit of time for an element to complete the work required for a single quorum access. Now consider a period of k time units, and some schedule of quorum accesses that need to take place during this period. Let the integers r_j count the number of times that each quorum $S_j \in \mathcal{S}$ is accessed, with the total number of accesses being $a = \sum_{S_j \in \mathcal{S}} r_j$. Ignoring the order in which the quorum accesses are scheduled, a necessary condition for the system to handle all a accesses within this period of k time units is that every element $i \in U$ be accessed at most k times. The following definition formalizes this condition using an integer linear program.

DEFINITION 3.7. *The maximum number of quorum accesses which a quorum system $\mathcal{S}$ can handle within k units of time is*

$$IP: a(\mathcal{S},k) = \max \sum_{j=1}^{m} r_j, \qquad s.t. \begin{cases} \sum_{S_j \ni i} r_j \le k, & \text{for all } i \in U, \\ r_j \ge 0, \\ r_j \in \mathbb{N}. \end{cases}$$

The capacity of the system $\mathcal{S}$ is defined as the maximal rate at which the system handles quorum accesses. In other words, the capacity is the number of accesses $a(\mathcal{S},k)$ that the system can handle, normalized by k. Since we are interested in the behavior over long time periods, we let the period k tend to infinity.

DEFINITION 3.8. *The* capacity *of a quorum system $\mathcal{S}$ is*

$$Cap(\mathcal{S}) = \lim_{k\to\infty} \frac{a(\mathcal{S},k)}{k}.$$

In hypergraph theory the quantity $a(\mathcal{S},k)$ is known as the *k-matching number* of $\mathcal{S}$ and is usually denoted by ν_k (cf. [12, p. 154]). Furthermore, Proposition 5.12 of [12] shows that $\lim_{k\to\infty} \nu_k/k = \nu^*$; hence by the definition of the capacity and Lemma 3.5 we obtain the following corollary.

COROLLARY 3.9. *Cap(S)= 1/$\mathcal{L}(\mathcal{S})$.*

Therefore, all the information regarding the capacity is captured by $\mathcal{L}(\mathcal{S})$. In [33] we gave a direct proof of Corollary 3.9 (without using the hypergraph machinery) which indicates how to schedule the quorum accesses so the capacity tends to $1/\mathcal{L}(\mathcal{S})$: select the quorums independently at random using a strategy w which optimizes the load.

3.4. The load with failures. In this section we extend our definition of the load to the case where the elements may fail. We use the simple probabilistic failure model of section 2.2, namely that the elements fail independently with probabilities $\mathbf{p} = (p_1, \ldots, p_n)$.

DEFINITION 3.10. *Let $\mathbf{x} \in \{0,1\}^n$ be the current configuration. Then $\mathcal{S}_\mathbf{x}$ is the subcollection of functioning quorums, $\mathcal{S}_\mathbf{x} = \{S \in \mathcal{S} : S \subseteq live(\mathbf{x})\}$.*

DEFINITION 3.11. *The load of a quorum system $\mathcal{S}$ over a configuration $\mathbf{x} \in \{0,1\}^n$ is defined as follows. If $\mathcal{S}_\mathbf{x} = \varnothing$ then $\mathcal{L}(\mathcal{S}_\mathbf{x}) = 1$. If there are functioning quorums, i.e., $\mathcal{S}_\mathbf{x} \neq \varnothing$, then*

$$\mathcal{L}(\mathcal{S}_\mathbf{x}) = \min L, \qquad s.t. \begin{cases} \sum_{S_j \in \mathcal{S}_\mathbf{x}} w_j = 1, \\ \sum_{\mathcal{S}_\mathbf{x} \ni S_j \ni i} w_j \le L, & \text{for all } i \in live(\mathbf{x}), \\ w_j \ge 0,\ L \ge 0. \end{cases}$$

Remark. When there are no functioning quorums in the current configuration, there is no natural concept of load. We choose to define $\mathcal{L}(\mathcal{S}_\mathbf{x}) = 1$ for such a configuration to capture the intuition of a monotonic load; as more elements fail, the load increases. The intuition behind this definition is justified in Proposition 3.16.

DEFINITION 3.12. *Let the elements fail with probabilities $\mathbf{p} = (p_1, \ldots, p_n)$. Then the load is a random variable $\mathcal{L}_\mathbf{p}(\mathcal{S})$ defined by*

$$\mathbb{P}\Big(\mathcal{L}_\mathbf{p}(\mathcal{S}) = L\Big) = \sum_{\substack{\mathbf{x} \\ \mathcal{L}(\mathcal{S}_\mathbf{x})=L}} \prod_{i \in dead(\mathbf{x})} p_i \prod_{i \in live(\mathbf{x})} q_i.$$

If the probabilities $\mathbf{p} = (p, \ldots, p)$ *are all equal, we denote the random load by* $\mathcal{L}_p(\mathcal{S})$. *Let* $E\mathcal{L}_p(\mathcal{S})$ *denote the expectation of* $\mathcal{L}_p(\mathcal{S})$.

FACT 3.13. *For any quorum system* $\mathcal{S}$, *if the elements never fail, then* $E\mathcal{L}_0(\mathcal{S}) = \mathcal{L}(\mathcal{S})$ *and if the elements fail with probability* 1, *then* $E\mathcal{L}_1(\mathcal{S}) = 1$.

LEMMA 3.14. *Let* $\mathcal{S}$ *be a quorum system. Then* $E\mathcal{L}_p(\mathcal{S}) \geq F_p(\mathcal{S})$ *for any* $p \in [0, 1]$.

Proof. By Definition 3.11, in a configuration $\mathbf{x}$ that causes a system failure (i.e., all the quorums are hit) the load is 1. Since $F_p(\mathcal{S})$ is the probability of a system failure, we get

$$E\mathcal{L}_p(\mathcal{S}) = [1 - F_p(\mathcal{S})] \cdot g(\mathcal{S}, p) + F_p(\mathcal{S}) \cdot 1$$

for some $g(\mathcal{S}, p) \geq 0$, and we are done. □

The following examples show that although the FPP quorum system and the Grid system have optimal or near optimal load of $O(1/\sqrt{n})$ when all the elements are functioning (see Example 4.11), this load is not *stable*.

Example 3.15. In [35] it is shown that $F_p(\text{FPP}) \underset{n\to\infty}{\longrightarrow} 1$ and $F_p(\text{Grid}) \underset{n\to\infty}{\longrightarrow} 1$ for any $p > 0$. Therefore Lemma 3.14 gives that $E\mathcal{L}_p(\mathcal{S}) \underset{n\to\infty}{\longrightarrow} 1$ for both systems. □

The next proposition shows the correctness of the intuition that if the elements are more fail prone, then the load is higher. For the proof we need some notation and two lemmas.

PROPOSITION 3.16. $E\mathcal{L}_p(\mathcal{S})$ *is a monotone nondecreasing function of* $p \in [0, 1]$ *for any* $\mathcal{S}$.

Notation. For configurations $\mathbf{x}$ and $\mathbf{z}$, denote $\mathbf{x} \geq \mathbf{z}$ if $x_i \geq z_i$ for all $i \in U$.

Notation. For a vector $\mathbf{z} = (z_1, \ldots, z_n)$, let $(1_i; \mathbf{z})$ denote the vector $\mathbf{z}$ with a 1 plugged into the ith coordinate: $(1_i; \mathbf{z}) = (z_1, \ldots, z_{i-1}, 1, z_{i+1}, \ldots, z_n)$, and similarly for $(0_i; \mathbf{z})$.

LEMMA 3.17. *Consider the function* $L(\mathbf{x}) : \{0, 1\}^n \mapsto [0, 1]$ *defined by* $L(\mathbf{x}) = \mathcal{L}(\mathcal{S}_\mathbf{x})$ *for some quorum system* $\mathcal{S}$. *If* $\mathbf{x} \geq \mathbf{z}$ *then* $L(\mathbf{x}) \geq L(\mathbf{z})$.

Proof. If $\mathbf{x} \geq \mathbf{z}$ then every element that is functioning in configuration $\mathbf{x}$ (with $x_i = 0$) is also functioning in $\mathbf{z}$. Therefore $\mathcal{S}_\mathbf{x} \subseteq \mathcal{S}_\mathbf{z}$. If $\mathcal{S}_\mathbf{x} = \varnothing$, then by Definition 3.11 $L(\mathbf{x}) = 1$ and we are done. Otherwise, any strategy w that uses only quorums of $\mathcal{S}_\mathbf{x}$ is a valid strategy for $\mathcal{S}_\mathbf{z}$ as well, and by the minimality of $\mathcal{L}(\mathcal{S}_\mathbf{z})$ the claim follows. □

LEMMA 3.18. *Let* $\mathcal{S}$ *be a quorum system, let the elements fail with probabilities* $\mathbf{p} = (p_1, \ldots, p_n)$, *and let* $L(\mathbf{x}) = \mathcal{L}(\mathcal{S}_\mathbf{x})$ *be the load over configuration* $\mathbf{x}$. *Consider the multilinear function* $h(\mathbf{p}) : [0, 1]^n \mapsto [0, 1]$ *defined by*

$$h(\mathbf{p}) = \sum_{\mathbf{x}\in\{0,1\}^n} L(\mathbf{x}) \prod_{x_k=1} p_k \prod_{x_k=0} q_k = \mathbb{E}[L(\mathbf{x})].$$

Then $\frac{\partial h}{\partial p_i} = h(1_i; \mathbf{p}) - h(0_i; \mathbf{p}) = \mathbb{E}[L(1_i; \mathbf{x}) - L(0_i; \mathbf{x})]$.

Proof. Sum $h(\mathbf{p})$ separately for configurations in which element i is failed ($x_i = 1$) or is functioning ($x_i = 0$).

$$\begin{aligned} h(\mathbf{p}) &= p_i \sum_{\mathbf{x}:x_i=1} L(\mathbf{x}) \prod_{\substack{x_k=1\\k\neq i}} p_k \prod_{x_k=0} q_k + q_i \sum_{\mathbf{x}:x_i=0} L(\mathbf{x}) \prod_{x_k=1} p_k \prod_{\substack{x_k=0\\k\neq i}} q_k \\ &= p_i h(p_1, \ldots, p_{i-1}, 1, p_{i+1}, \ldots, p_n) + q_i h(p_1, \ldots, p_{i-1}, 0, p_{i+1}, \ldots, p_n) \\ &= p_i h(1_i; \mathbf{p}) + q_i h(0_i; \mathbf{p}). \end{aligned}$$

Taking partial derivatives we get $\frac{\partial h}{\partial p_i} = h(1_i; \mathbf{p}) - h(0_i; \mathbf{p})$. Having element i fail with probability 1 is the same as having a constant 1 in the random configuration $\mathbf{x}$, so $h(1_i; \mathbf{p}) = \mathbb{E}[L(1_i; \mathbf{x})]$. Linearity of the expectation completes the lemma. □

Proof of Proposition 3.16. Consider the case where the elements fail with probabilities $\mathbf{p} = (p_1, \ldots, p_n)$, and let $L(\mathbf{x})$ and $h(\mathbf{p})$ be as before. By Lemma 3.17 $L(\mathbf{x})$ is nondecreasing, so $L(1_i; \mathbf{x}) \geq L(0_i; \mathbf{x})$ for every i. Therefore by Lemma 3.18, $\frac{\partial h}{\partial p_i} \geq 0$ as an expectation of nonnegative terms, so $h(\mathbf{p})$ is nondecreasing in every coordinate. Plugging $\mathbf{p} = (p, \ldots, p)$ shows that $E\mathcal{L}_p(\mathcal{S}) = h(p, \ldots, p)$ is a nondecreasing function. □

3.5. Other measures of load. In order to measure an intuitive notion of "load" of a quorum system, our definition of $\mathcal{L}(\mathcal{S})$ (Definitions 3.2 and 3.3) is not the only one that comes to mind. Here we discuss the shortcomings of some alternatives.

Several authors (e.g., [28, 1] have emphasized the criterion of having small quorums. This is an important parameter since it captures the message complexity of a protocol using the quorum system. However, it does not tell us *how to use* the quorums so each element is used as infrequently as possible. Moreover, our lower bounds (Propositions 4.1 and 4.2) show that if the quorum size is small (i.e., $c(\mathcal{S}) < \sqrt{n}$) then decreasing it any further actually increases the load. We therefore argue that when analyzing a quorum system, one should consider *both* its quorum size and load (and of course its availability) since each measures a different aspect of the system's quality. Having a small quorum size does not give us the whole picture.

Looking for systems with small average quorum size can also be misleading. For instance, the average quorum size in the Wheel system [29] is very small (≈ 3) but the load is high: $\mathcal{L}(\text{Wheel}) \approx 1/2$.

Another tempting definition is that of an average load, rather than the maximum, i.e., $AvL(\mathcal{S}) = \min_w \frac{1}{n} \sum_{i \in U} \sum_{S_j \ni i} w_j$, minimizing over strategies w. An equivalent notion is that of the total load, which is the same as the average except for the scaling factor of $1/n$. However, by changing the summation order it follows that $AvL(\mathcal{S}) = \min_w \frac{1}{n} \sum_{1 \leq j \leq m} w_j |S_j|$. A strategy that minimizes this expression is the trivial strategy that always uses the smallest quorum $S_{\min}$ (with probability 1), so AvL is an uninteresting measure.

4. Properties of the load.

4.1. Lower bounds and a tradeoff of the load. In this section we present three lower bounds on the load $\mathcal{L}(\mathcal{S})$, in terms of the smallest quorum cardinality $c(\mathcal{S})$ and the universe size n. Two of these can be found in the hypergraph literature as upper bounds for the fractional matching number ν^*, and we present them here using our terminology. We also show a tradeoff between the availability of a quorum system, quantified by the failure probability F_p, and the load.

PROPOSITION 4.1. (See [12, p. 150].) $\mathcal{L}(\mathcal{S}) \geq \frac{c(\mathcal{S})}{n}$ *for any quorum system* $\mathcal{S}$.

PROPOSITION 4.2. $\mathcal{L}(\mathcal{S}) \geq \frac{1}{c(\mathcal{S})}$ *for any quorum system* $\mathcal{S}$.

Proof. Let $S_{\min} \in \mathcal{S}$ be a quorum such that $|S_{\min}| = c(\mathcal{S})$ and let $\mathbf{y}$ be defined by $y_i = \frac{1}{c(\mathcal{S})}$ for $i \in S_{\min}$ and $y_i = 0$ otherwise. Then $(\mathbf{y}; 1/c(\mathcal{S}))$ is feasible for program DLP so the claim follows by the weak duality of linear programming. □

PROPOSITION 4.3. (See [2]; cf. [12, p. 170].) *Let* $m(\mathcal{S})$ *be the number of quorums in* $\mathcal{S}$. *Then*

$$\mathcal{L}(\mathcal{S}) \geq \frac{1}{\sqrt{n}} \sqrt{1 + \frac{c(\mathcal{S}) - 1}{m(\mathcal{S})}} \geq \frac{1}{\sqrt{n}}.$$

Example 4.4. The following examples show that both Propositions 4.1 and 4.2 give meaningful lower bounds on the load of some quorum systems.

(i) Over an odd-sized universe, all the quorums of the simple majority system Maj are of size $(n+1)/2$; therefore, by Proposition 4.1 $\mathcal{L}(\text{Maj}) \geq (n+1)/2n > \frac{1}{2}$.

(ii) In the Tree system [1], the smallest quorums have cardinality $\log(n+1)$. Therefore, by Proposition 4.2 $\mathcal{L}(\text{Tree}) \geq 1/\log(n+1)$. □

The following proposition shows a tradeoff between the failure probability and the load.

PROPOSITION 4.5. *$F_p(\mathcal{S}) \geq p^{n\mathcal{L}(\mathcal{S})}$ for any quorum system $\mathcal{S}$ and any $p \in [0,1]$.*

Proof. Consider a quorum $S_{\min}$ with $|S_{\min}| = c(\mathcal{S})$. If all the elements of $S_{\min}$ fail then by the intersection property the system fails; therefore $F_p(\mathcal{S}) \geq p^{c(\mathcal{S})}$. The claim follows since $c(\mathcal{S}) \leq n\mathcal{L}(\mathcal{S})$ by Proposition 4.1. □

DEFINITION 4.6. *An infinite family of quorum systems $\mathcal{S}_n$ over universes of increasing size n is said to have a* tight tradeoff *if*

$$\mathcal{L}(\mathcal{S}) \leq C \cdot \frac{-\log F_p(\mathcal{S}_n)}{n}$$

for some constant $C = C(p) > 0$ that depends only on $0 < p < \frac{1}{2}$.

Remark. It is pointless to consider values of $p \geq \frac{1}{2}$ since in [35] it is proved that $F_p(\mathcal{S}) \geq \frac{1}{2}$ for such p and any quorum system $\mathcal{S}$, so Proposition 4.5 is meaningless asymptotically in this case.

4.2. Conditions for optimality of the load. In this section we present several conditions that guarantee the optimality of a strategy w. The first condition, which can be applied to any system $\mathcal{S}$, is an immediate consequence of linear programming duality.

PROPOSITION 4.7. *Let a quorum system $\mathcal{S}$ be given, and let w be a strategy for $\mathcal{S}$ with an induced load of $\mathcal{L}_w(\mathcal{S}) = L$. Then L is the optimal load iff there exists $\mathbf{y} \in [0,1]^n$ such that $\mathbf{y}(U) = 1$ and $\mathbf{y}(S) \geq L$ for all $S \in \mathcal{S}$.*

Proof. By the premise, $(w; L)$ is a feasible point of LP, with an objective function value of L. Therefore, by duality L is the optimum iff there exists a feasible point of the dual problem DLP with an objective function value of L as well. By the definition of DLP, this implies that L is optimal iff there exists $\mathbf{y}$ such that $(\mathbf{y}; L)$ is dual-feasible, which is guaranteed by the conditions on $\mathbf{y}$. □

One way to search for a good strategy w is to try to find a *balancing strategy.* We can try to do this by constructing a feasible point $(w; L)$ for the following *balanced* load linear program, in which the inequalities (ii) of LP are replaced by equalities (ix).

$$BLP: \begin{cases} \sum_{j=1}^{m} w_j = 1, & \text{(viii)} \\ \sum_{S_j \ni i} w_j = L, \quad \text{for all } i \in U, & \text{(ix)} \\ w_j \geq 0,\ L \geq 0. & \text{(x)} \end{cases}$$

The program BLP is not always feasible, since finding a solution would imply that $\mathcal{S}$ can be *perfectly balanced*, which cannot be done for all quorum systems [19]. Nevertheless, one could hope that such a balancing strategy (if found) would induce the optimal load. The next proposition shows that this is true for a certain subclass of quorum systems.

PROPOSITION 4.8. *Let $\mathcal{S}$ be an s-uniform quorum system. Let w be a strategy and let $L \geq 0$ be such that $(w; L)$ is a feasible point for program BLP. Then the optimal load is $\mathcal{L}(\mathcal{S}) = L = s/n$.*

Proof. First let us show that $L = s/n$. Using the equalities (ix) we get

$$\sum_{i \in U} \sum_{S_j \ni i} w_j = nL. \tag{1}$$

By switching the summation order and using the s-uniformity of $\mathcal{S}$ and equality (viii) we get

$$\sum_{i \in U} \sum_{S_j \ni i} w_j = \sum_{j=1}^{m} w_j \sum_{i \in S_j} 1 = s \sum_{j=1}^{m} w_j = s. \tag{2}$$

By equating (1) and (2) we conclude that $L = s/n$.

Now let $\mathbf{y} = (1/n, \ldots, 1/n)$ be a weight vector for the elements. Clearly $\mathbf{y}(U) = 1$, and $\mathbf{y}(S) = |S|/n = s/n = L$ for any quorum $S \in \mathcal{S}$, since $\mathcal{S}$ is s-uniform. Therefore $(\mathbf{y}; L)$ is dual-feasible, so by Proposition 4.7, $\mathcal{L}(\mathcal{S}) = L$. □

Remark. The proof does not use the fact that $\mathcal{S}$ is a *quorum system* in any way, and it holds for nonintersecting set systems as well.

The condition that Proposition 4.8 places on a strategy w is a very weak one. It suffices to show that w is a feasible balancing strategy for it to induce the unique optimal load, if $\mathcal{S}$ is uniform. The following example shows that the uniformity is crucial; nonuniform quorum systems can have several balancing strategies, with different induced loads.

Example 4.9. Consider the quorum system

$$\mathcal{S} = \{\ \{1,4,6\}, \{2,4,7\}, \{3,5,6,7\}, \{1,2,3,5\},$$
$$\{1,2,3,4\}, \{2,3,4,5\}, \{3,4,5,6\}, \{4,5,6,7\}, \{5,6,7,1\}, \{6,7,1,2\}, \{7,1,2,3\}\ \}.$$

The strategy $w = (0, 0, 0, 0, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7})$ is balancing with a load of $\mathcal{L}_w(\mathcal{S}) = \frac{4}{7}$. However, the strategy $w' = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, 0, 0, 0, 0, 0, 0, 0)$ is also balancing, with a load of $\mathcal{L}_{w'}(\mathcal{S}) = \frac{1}{2}$. □

If $\mathcal{S}$ is a fair system, then the next proposition shows that we can compute the load and optimal strategy immediately. This is a restatement of Proposition 5.1 of [12] using the fact that $\mathcal{L}(\mathcal{S}) = 1/\nu^*$.

PROPOSITION 4.10. *Let $\mathcal{S}$ be an (s, d)-fair quorum system. Then $\mathcal{L}(\mathcal{S}) = s/n = d/m$.*

Example 4.11. The following examples demonstrate the use of Proposition 4.10. The first shows that the lower bound of Example 4.4 is tight, and the other two show that the optimal load of Proposition 4.3, $1/\sqrt{n}$, can be attained.

(i) Over an odd-sized universe, Maj is an $\frac{n+1}{2}$-fair quorum system, so $\mathcal{L}(\text{Maj}) = \frac{n+1}{2n} \approx \frac{1}{2}$.

(ii) The FPP system [28] is a $(t+1)$-fair quorum system over $n = t^2 + t + 1$ elements, so $\mathcal{L}(\text{FPP}) = \frac{t+1}{t^2+t+1} \approx \frac{1}{\sqrt{n}}$. In fact, equality holds in the tighter lower bound of Proposition 4.3 for this system.

(iii) The Grid system [5] is a $(2h-1)$-fair system over $n = h^2$ elements, so $\mathcal{L}(\text{Grid}) = \frac{2h-1}{h^2} \approx \frac{2}{\sqrt{n}}$.

4.3. Effective calculation of the load. If a quorum system $\mathcal{S}$ is given explicitly, as a list of all m quorums, then program LP of Definition 3.3 can be solved in $poly(n, m)$ time using linear programming (cf. [40]). However, often $\mathcal{S}$ is given implicitly, say, via some data structure containing the elements coupled with a polynomial-time procedure to produce a quorum on demand. In such a case just writing program

```
Input a point (y; T).
The rows are U_1, ..., U_d.
Q ← ∅; s ← 0
for i = d to 1                          // bottom to top
    r ← Σ_{j∈U_i} y_j
    if r + s < T then
        return U_i ∪ Q                  // y(U_i ∪ Q) < T
    else
        j ← argmin_{k∈U_i}{y_k}         // min weight in row i
        s ← s + y_j
        Q ← Q ∪ {j}
end-for
return TRUE                             // (y; T) is dual-feasible
```

FIG. 1. *An oracle for a crumbling wall quorum system.*

LP could be an exponential task since typically $m = 2^{\Omega(n)}$. Calculating the load quickly is especially important when failures may occur, since the computation needs to be done repeatedly after each configuration change.

Instead we make use of the adaptation of the ellipsoid algorithm of [16]. Let d denote the dimension of the problem at hand. The ellipsoid algorithm uses an oracle, which receives a point $x \in \mathbb{R}^d$ and performs the following action.

(i) If x is a feasible point, then return TRUE.

(ii) Otherwise, return a hyperplane separating x from the feasible region (i.e., return a violated constraint).

Given such an oracle that works in time τ, the algorithm solves the linear program in time $poly(\tau, d)$.

We achieve nothing by applying this algorithm to problem LP since its dimension is $m + 1$. However, we can apply this algorithm to the dual problem DLP, whose dimension is $n+1$. Translated to our terminology, we need to provide an oracle whose input is a point $(\mathbf{y}; T)$. If $(\mathbf{y}; T)$ is feasible in DLP then the oracle returns TRUE, otherwise it returns a quorum $S \in \mathcal{S}$ such that $\mathbf{y}(S) < T$. If this oracle works in $poly(n)$ time, then the algorithm calculates the load in $poly(n)$ time.

Remark. Solving problem DLP gives us the optimal value of the load, but does *not* find a strategy that induces this load. Just writing down a strategy would cause a time complexity of $\Omega(m)$.

Example 4.12. In the systems of the crumbling wall class [37] the elements are arranged in rows of different widths, and a quorum is the union of a full row and a representative from each row below the full row. The procedure in Figure 1 is an oracle of the required kind, with a time complexity of $O(n)$. Therefore, we can compute the load of any crumbling wall using the ellipsoid algorithm outlined above. □

5. Optimal load, high availability quorum systems.

5.1. The paths system. In this system, the elements constitute a type of square grid, and a quorum is the union of two paths, one connecting the left and right sides and one connecting the top and bottom sides. Our analysis shows that $\mathcal{L}(\text{Paths}) = O(\frac{1}{\sqrt{n}})$ and that $F_p(\text{Paths}) \leq e^{-\Omega(\sqrt{n})}$ for $p < \frac{1}{2}$, so the tradeoff between the load

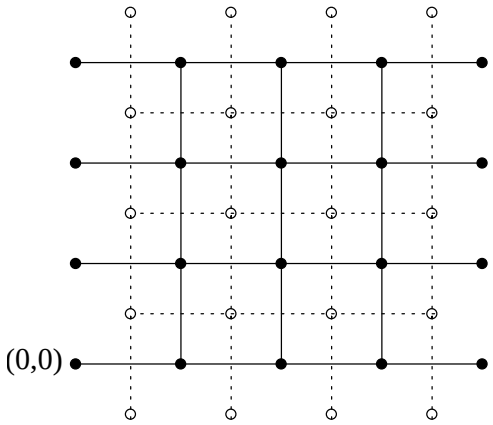


FIG. 2. *The grids* $G(3)$ (*full lines*) *and* $G^*(3)$ (*dotted lines*).

and failure probability is tight. Moreover, we show that even in the presence of faults, with exponentially high probability the load is still $\mathcal{L}_p(\text{Paths}) = O(\frac{1}{\sqrt{n}})$ for all $p < \frac{1}{2}$, which is the best we can hope for. We also give a simple and efficient algorithm for computing a strategy which induces an almost optimal load when some elements are faulty. The proofs are based on theorems of percolation theory (see the appendix).

DEFINITION 5.1. *Let* $G(d)$ *be the subgrid of* $\mathbb{Z}^2$ *with vertex set* $\{v \in \mathbb{Z}^2 : 0 \le v_1 \le d+1, 0 \le v_2 \le d\}$ *and edge set consisting of all edges joining neighboring vertices* except *those joining vertices* u, v *with either* $u_1 = v_1 = 0$ *or* $u_1 = v_1 = d+1$.

DEFINITION 5.2. *Let* $G^*(d)$, *the dual of* $G(d)$, *be the subgrid with vertex set* $\{v + (\frac{1}{2}, \frac{1}{2}) : 0 \le v_1 \le d, -1 \le v_2 \le d\}$ *and edge set consisting of all edges joining neighboring vertices* except *those joining vertices* u, v *with either* $u_2 = v_2 = -\frac{1}{2}$ *or* $u_2 = v_2 = d + \frac{1}{2}$.

See Figure 2 for a drawing of $G(d)$ and $G^*(d)$. Note that every edge $e \in G(d)$ has a dual edge $e^* \in G^*(d)$ which *crosses* it. We call such e and e^* a *dual pair of edges*. Note also that $G(d)$ and $G^*(d)$ are isomorphic; $G^*(d)$ may be obtained by rotating $G(d)$ at a right angle around the origin and relocating the vertex labeled $(0,0)$ to the point $(d + \frac{1}{2}, -\frac{1}{2})$. Both $G(d)$ and $G^*(d)$ contain $d^2 + (d+1)^2 = 2d^2 + 2d + 1$ edges.

DEFINITION 5.3. *The* Paths *quorum system of order* d *has* $n = 2d^2 + 2d + 1$ *elements, and we identify an element in* U *with a dual pair of edges* $e \in G(d)$ *and* $e^* \in G^*(d)$. *A quorum in the system is the union of (elements identified with) the edges of a left-right path in* $G(d)$ *and the edges of a top-bottom path in* $G^*(d)$.

PROPOSITION 5.4. $\frac{\sqrt{2}}{\sqrt{n}} \lessapprox \mathcal{L}(\text{Paths}) \lessapprox \frac{2\sqrt{2}}{\sqrt{n}}$.

Proof. For the lower bound, note that the smallest quorum has size $c(\text{Paths}) = 2d+1$, and we can apply Proposition 4.1 to get $\mathcal{L}(\text{Paths}) \ge \frac{2d+1}{2d^2+2d+1}$. For the upper bound, consider the quorums of the type $S_j = \{\text{edges joining } u, v \in G(d) : u_2 = v_2 = j\} \cup \{\text{edges joining } u, v \in G^*(d) : u_1 = v_1 = j + \frac{1}{2}\}$, for $j = 0, \ldots, d$. Each element corresponding to a horizontal edge in $G(d)$ appears in two such quorums, except elements on the diagonal that appear only once. A strategy choosing only these quorums with probability $\frac{1}{d+1}$ induces a load of $\frac{2}{d+1}$. □

We now wish to calculate the failure probability of the Paths system. We assume that the elements fail with probability p. A failed element corresponds to *two* closed

percolation edges: an edge $e \in G(d)$ and its dual edge $e^* \in G^*(d)$. We say that a *path in $G(d)$ is closed* if *all* its edges are closed. Define the following events:

(i) $LR =$ "there exists an open left-right path in $G(d)$,"
(ii) $LRC =$ "there exists a closed left-right path in $G(d)$,"
(iii) $TB =$ "there exists a open top-bottom path in $G^*(d)$,"
(iv) $TBC =$ "there exists a closed top-bottom path in $G^*(d)$."

LEMMA 5.5. *If $p > \frac{1}{2}$, then there exists a positive function φ such that $\mathbb{P}_p(LR) \le e^{-\varphi(p)d}$.*

Proof. Consider the grid $G(d)$, and let $\lambda = \{v \in \mathbb{Z}^2 : v_1 = d+1\}$ be the set of $\mathbb{Z}^2$ vertices on the infinite vertical line on the right side of $G(d)$. Let R denote the vertices on the right side of $G(d)$. Then summing along the possible starting points on the left side,

$$\mathbb{P}_p(LR) \le \sum_{k=0}^{d} \mathbb{P}_p((0,k) \leftrightarrow R) \le \sum_{k=0}^{d} \mathbb{P}_p((0,k) \leftrightarrow \lambda) = (d+1)\mathbb{P}_p(0 \leftrightarrow \lambda).$$

A path from the origin to λ must exit the ball $B(d)$, so we can apply Theorem A.1 to get

$$\le (d+1)\mathbb{P}_p(0 \leftrightarrow \partial B(d)) \le (d+1)e^{-\psi(p)d} \le e^{-\varphi(p)d}. \qquad \square$$

COROLLARY 5.6. *If $q > \frac{1}{2}$ ($p < \frac{1}{2}$), then there exists a positive function φ such that $\mathbb{P}_p(LRC) \le e^{-\varphi(q)d}$.*

Proof. Exchanging the roles of p and q we get that $\mathbb{P}_p(LRC) = \mathbb{P}_q(LR)$, so we can apply Lemma 5.5. $\square$

PROPOSITION 5.7. *There exists a positive function φ such that $F_p(\text{Paths})$ obeys*

$$\begin{cases} F_p(\text{Paths}) \le 2e^{-\varphi(q)d}, & p < \frac{1}{2}, \\ F_p(\text{Paths}) \ge 1 - e^{-\varphi(p)d}, & p > \frac{1}{2}, \\ \frac{1}{2} < F_p(\text{Paths}) \le \frac{3}{4}, & p = \frac{1}{2}, \end{cases}$$

so $F_p(\text{Paths})$ is almost Condorcet.

Proof. By definition, the event "there is a live quorum" is $LR \cap TB$. A moment's reflection shows that an open left-right path exists in $G(d)$ iff no closed top-bottom path exists in $G^*(d)$, since a dual pair of edges e and e^* have the same state (see discussion in [15, pp. 198–199]). Therefore, the events LR and TBC are complementary. Since $G(d)$ and $G^*(d)$ are isomorphic, then TB and LRC are also complementary events. Therefore, the system failure event is

$$\mathit{fail} = \overline{LR \cap TB} = TBC \cup LRC.$$

Additionally, the isomorphism between $G(d)$ and $G^*(d)$ implies that $\mathbb{P}_p(LR) = \mathbb{P}_p(TB)$ and $\mathbb{P}_p(LRC) = \mathbb{P}_p(TBC)$. Now we consider the three cases as follows.

(i) Let $p < \frac{1}{2}$. Then $F_p = \mathbb{P}_p(\mathit{fail}) = \mathbb{P}_p(LRC \cup TBC) \le 2\mathbb{P}_p(LRC)$ and so $F_p(\text{Paths}) \le 2e^{-\varphi(q)d}$ by Corollary 5.6.

(ii) Let $p > \frac{1}{2}$. Then $1 - F_p = \mathbb{P}_p(LR \cap TB) \le \mathbb{P}_p(LR) \le e^{-\varphi(p)d}$ by Lemma 5.5.

(iii) Let $p = \frac{1}{2}$. From the above discussion and the proof of Corollary 5.6 it follows that $\mathbb{P}_{1/2}(LR) = \mathbb{P}_{1/2}(TB) = \frac{1}{2}$. For the upper bound, note that both LR and TB are increasing events, so we can use the FKG inequality [10]. Therefore,

$$F_{1/2} = 1 - \mathbb{P}_{1/2}(LR \cap TB) \le 1 - \mathbb{P}_{1/2}(LR)\mathbb{P}_{1/2}(TB) = \frac{3}{4}.$$

For the lower bound, note that Paths is a *dominated* quorum system. Therefore, $F_{1/2}(\text{Paths}) > \frac{1}{2}$ by a result of [35]. □

Finally, we show that, with high probability, the load of the Paths system is $O(\frac{1}{\sqrt{n}})$ in the presence of failures, for any failure probability $p < \frac{1}{2}$. In other words, the load has essentially the same asymptotic behavior as long as there is a good probability that at least one functioning quorum exists.

PROPOSITION 5.8. *For any* $0 \le p < \frac{1}{2}$ *there exists* $\gamma > 0$ *such that* $\mathcal{L}_p(\text{Paths}) = O(\frac{1}{\sqrt{n}})$ *with probability* $\ge 1 - e^{-\gamma d}$.

Proof. Let LR_r be the event "there exist at least $r+1$ edge disjoint left-right paths in $G(d)$." Fix some $\frac{1}{2} > p' > p$. Then by Theorem A.3,

$$1 - \mathbb{P}_p(LR_r) \le \left(\frac{q}{q-q'}\right)^r [1 - \mathbb{P}_{p'}(LR)].$$

Now for $p' < \frac{1}{2}$,

$$\mathbb{P}_{p'}(LR) = 1 - \mathbb{P}_{p'}(TBC) = 1 - \mathbb{P}_{p'}(LRC) \ge 1 - e^{-\varphi(q')d}$$

by Corollary 5.6, so

$$\mathbb{P}_p(LR_r) \ge 1 - \left(\frac{q}{q-q'}\right)^r e^{-\varphi(q')d}.$$

Fix $0 < \gamma < \varphi$, let $0 < \beta = \frac{\varphi-\gamma}{\ln[q/(q-q')]}$, and let $r = \beta d$. Then $\mathbb{P}_p(LR_r) \ge 1 - e^{-\gamma d}$. In other words, with high probability, there exist $\beta d + 1$ edge disjoint left-right paths in $G(d)$. The same also happens for top-bottom paths in $G^*(d)$. Therefore, we can find $\beta d + 1$ quorums such that any element appears in at most two of them (once as an edge $e \in G(d)$ and once as the dual edge). We conclude that when such quorums are found, $\mathcal{L}_p(\text{Paths}) \le \frac{2}{\beta d+1} = O(\frac{1}{\sqrt{n}})$. □

Remark. This is the strongest possible result regarding load with failures, since if $p \ge \frac{1}{2}$ then by Lemma 3.14 and a result of [35], $E\mathcal{L}_p(\mathcal{S}) \ge F_p(\mathcal{S}) \ge \frac{1}{2}$ for any quorum system $\mathcal{S}$.

Proposition 5.8 guarantees that, with high probability, a good strategy (that induces a load of $O(1/\sqrt{n})$) exists. We now describe an efficient algorithm that finds a nearly optimal strategy w for any given configuration $\mathbf{x}$; the load induced by w is at most twice the optimal load under configuration $\mathbf{x}$, $\mathcal{L}(\text{Paths}_\mathbf{x})$.

The algorithm mimics the structure of the existence proof. As a preprocessing step that needs to be performed after each configuration change, the algorithm finds a maximum collection of disjoint left-right paths, say k_{LR} such paths, and similarly finds k_{TB} disjoint top-bottom paths. This can be done by connecting a source vertex s to all the vertices on the left side and a sink t to the vertices on the right, assigning a capacity of 1 to all the edges, and finding the maximum (s,t) flow (and repeating for TB paths). Since the network is planar we can find the flow in time $O(n \log n)$ using the algorithm of [20], or in time $O(n\sqrt{\log n})$ by [17] using the methods of [11].

Given these path collections, the strategy w is the following: if either $k_{LR} = 0$ or $k_{TB} = 0$, then no live quorums exist in configuration $\mathbf{x}$. Otherwise, whenever a quorum is needed, pick an LR path with uniform probability $1/k_{LR}$ and a TB path with uniform probability $1/k_{TB}$, and use their union. Since the paths are disjoint, each element can appear at most once in an LR path and once in a TB path, so

$$\mathcal{L}_w(\text{Paths}_\mathbf{x}) \le 1/k_{LR} + 1/k_{TB}.$$

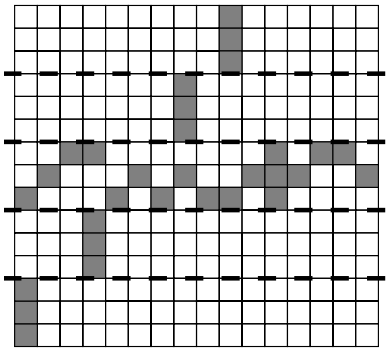


FIG. 3. *The* B-Grid *system over* $n = 240$ *elements with width* $d = 16$, $h = 5$ *bands, and* $r = 3$ *rows per band. One quorum is shaded.*

However, if the maximum flow is k_{LR}, then the max-flow min-cut theorem implies the existence of a k_{LR}-size cut. Therefore, any LR path that is open in configuration $\mathbf{x}$ must cross this cut via an edge, so some edge in this cut must have a load of at least $1/k_{LR}$ under any strategy. This implies a lower bound on the load

$$\mathcal{L}(\text{Paths}_{\mathbf{x}}) \geq \max\{1/k_{LR}, 1/k_{TB}\};$$

hence $\mathcal{L}_w(\text{Paths}_{\mathbf{x}})$ is at most twice the best possible.

Remark. A related construction, using paths on a triangular lattice with elements corresponding to the nodes, was suggested in [45] (see [44]). They show that their construction has asymptotically high availability ($F_p \to 0$ when $p < \frac{1}{2}$ in our notation). The rate of convergence is not analyzed and neither is the load (with or without failures). Nevertheless, it seems that an analysis similar to ours would show that the characteristics of their system are comparable to those of our Paths system, with a load of $O(1/\sqrt{n})$ and $F_p \leq e^{-\Omega(\sqrt{n})}$ when $p < \frac{1}{2}$.

5.2. The B-Grid system. Arrange the elements in a rectangular grid of width d. Split the grid logically into h *bands* of r rows each (so there are $n = dhr$ elements). Call r elements in a column that are all contained in a single band a *minicolumn*. Then a quorum consists of one minicolumn in every band, and a representative element in each minicolumn of one band (see Figure 3).

LEMMA 5.9. $\mathcal{L}(\text{B-Grid}) = \frac{d+hr-1}{dhr}$.

Proof. Clearly B-Grid is a fair quorum system, with a quorum size of $d + hr - 1$, and the lemma follows from Proposition 4.10. □

LEMMA 5.10. $F_p(\text{B-Grid}) \leq \left(dp^r\right)^h + h\left(1 - q^r\right)^d$.

Proof. Define $\mathcal{E}_1$ to be the event "in every band there exists a minicolumn whose elements all failed," and $\mathcal{E}_2$ to be the event "there exists a band in which every minicolumn contains a failed element." Clearly the system failure event is $fail = \mathcal{E}_1 \cup \mathcal{E}_2$, so $F_p(\text{B-Grid}) \leq \mathbb{P}(\mathcal{E}_1) + \mathbb{P}(\mathcal{E}_2)$. We get the result since $\mathbb{P}(\mathcal{E}_1) \leq \left(dp^r\right)^h$ and $\mathbb{P}(\mathcal{E}_2) \leq h\left(1 - q^r\right)^d$. □

In the next lemma we give a condition on r under which F_p decays exponentially fast in a large range of p values.

LEMMA 5.11. *If* $0 \leq p \leq \frac{1}{3}$ *and* $r = \lfloor \ln d \rfloor$, *then* $F_p(\text{B-Grid}) \leq e^{-h} + e^{-\frac{1}{2}\sqrt{d}}$ *for large values of* d *such that* $\ln h < \frac{1}{2}\sqrt{d}$.

AbC	bCD	CDE	DEA	EAb
fGh	GhI	hIj	Ijf	jfG
Klm	lmN	mNo	NoK	oKl

FIG. 4. *The* SC-Grid *system over* $n = 15$ *elements with width* $d = 5$, $h = 3$ *rows, and* $r = 3$ *elements per cell. The elements of one quorum are marked by capitalized letters, and the cells where a majority is achieved are shaded.*

Proof. To get $\mathbb{P}(\mathcal{E}_1) \le \left(dp^r\right)^h \le e^{-h}$ we require the condition

$$r > \frac{\ln d + 1}{\ln 1/p} \ . \tag{3}$$

To get $\mathbb{P}(\mathcal{E}_2) \le h\left(1-q^r\right)^d \le e^{-\frac{1}{2}\sqrt{d}}$ we require the condition

$$r < \frac{\ln d - \ln(\ln h + \frac{1}{2}\sqrt{d})}{\ln 1/q} \ . \tag{4}$$

If we consider only $p \le \frac{1}{3}$, then $\frac{1}{\ln 1/p} \le 0.91$ and $\frac{1}{\ln 1/q} \ge 2.466$, and a simple check shows that $r = \lfloor \ln d \rfloor$ fills both conditions (3) and (4) for sufficiently large d if $\ln h < \frac{1}{2}\sqrt{d}$. □

The next propositions are proved by plugging the parameters into Lemmas 5.11 and 5.9. In Proposition 5.12 the failure probability is minimal for the B-Grid system (up to a logarithmic factor in the exponent). In Proposition 5.13 the load is minimal.

PROPOSITION 5.12. *If* $d = n^{2/3}$, $r = \lfloor \ln d \rfloor$, *and* $h = n/(rd)$, *then* $\mathcal{L}(\text{B-Grid}) = O(n^{-1/3})$ *and* $F_p(\text{B-Grid}) = O(\exp(-\frac{n^{1/3}}{\ln n}))$ *in the range* $0 \le p \le \frac{1}{3}$.

PROPOSITION 5.13. *If* $d = \sqrt{n}$, $r = \lfloor \ln d \rfloor$, *and* $h = n/(rd)$, *then* $\mathcal{L}(\text{B-Grid}) = O(1/\sqrt{n})$ *and* $F_p(\text{B-Grid}) = O(\exp(-\frac{n^{1/4}}{2}))$ *in the range* $0 \le p \le \frac{1}{3}$.

Remark. Taking either $d > n^{2/3}$ or $d < \sqrt{n}$ makes both the load and the availability worse. Note that, in any case, the tradeoff between the load and failure probability is *not* tight. By Proposition 4.5 we could hope for a failure probability of $O(\exp(-n^{2/3}))$ when the load is $\approx n^{-1/3}$.

5.3. The SC-Grid system. Consider a grid made of h rows of cells with width d. In a universe of size $n = dh$, allocate d different elements to each row. Assume that row j is allocated elements $\{1, \ldots, d\}$. Then for a parameter $r < d$, place the elements into cells in shifted cyclic order: $\{1, \ldots, r\}$ in cell $(1, j)$, $\{2, \ldots, r+1\}$ in cell $(2, j)$ and so forth. Every element appears in r cells in the same row. A quorum in the system is a set of elements that are a majority in one cell of every row and a majority in every cell of one row (see Figure 4). This system is somewhat similar to that of [38] in which each grid cell contains a *distinct* set of elements. For simplicity assume that both d and r are odd.

LEMMA 5.14. *Let* r *be odd and let* $d > r$. *Consider a cycle of* d *elements, and the* d *subsets* $C_1, \ldots, C_d$ *of* r *consecutive elements along the cycle. Color* G *of the*

elements in green, and let g_j count the number of green elements in C_j. If $g_j \geq \frac{r+1}{2}$ for all j, then $G \geq \lceil d \cdot \frac{r+1}{2r} \rceil$. If $d \mid r$ then the bound can be achieved.

Proof. Sum the number of green elements in each C_j. Then $\sum_{j=1}^{d} g_j = rG$ since every green element is counted precisely r times. Since $g_j \geq \frac{r+1}{2}$ then $rG \geq d \cdot \frac{r+1}{2}$ and we are done.

If $d = rx$ for some integer x, then consider the x disjoint sets $C_\ell = \{(\ell-1)r + 1, \ldots, \ell r\}$ for $1 \leq \ell \leq x$. In each set color the first $\frac{r+1}{2}$ elements in green. Then *every* set C_j contains $\frac{r+1}{2}$ green elements and $G = x \cdot \frac{r+1}{2}$, so the lower bound is achieved. ∎

LEMMA 5.15. $\frac{rh+d}{2n} \lessapprox \mathcal{L}(\text{SC-Grid}) \lessapprox \frac{r}{d}$.

Proof. By Lemma 5.14 the smallest quorum size is $c(\text{SC-Grid}) \geq \frac{r+1}{2}(h-1) + \frac{d+1}{2}$, so the lower bound follows from Proposition 4.1. For the upper bound, consider the quorums $S_1, \ldots, S_{k+1}$, where S_j contains all the d elements of row j, and $\frac{r+1}{2}$ elements of every row $i \neq j$. Consider a specific row j. As long as $k\frac{r+1}{2} \leq d$ we can use a different set of $\frac{r+1}{2}$ elements from row j in quorum S_i for $i \neq j$, so every element appears in at most two quorums. Therefore we can take $k = \lfloor \frac{2d}{r+1} \rfloor$. A strategy that chooses one of these quorums with equal probability induces a load of $\frac{2}{k+1} \approx \frac{r}{d}$. ∎

Notation. Let f_x be the probability that at least $\frac{x+1}{2}$ elements fail out of x when each element fails independently with probability p.

LEMMA 5.16. $F_p(\text{SC-Grid}) \leq (df_r)^h + hf_d$.

Proof. Call a cell *failed* if a majority of its elements fail. Let $\mathcal{E}_1$ be the event "all the rows contain at least one failed cell," and let $\mathcal{E}_2$ be the event "there exists a row in which all the cells failed." Then $F_p(\text{SC-Grid}) \leq \mathbb{P}(\mathcal{E}_1) + \mathbb{P}(\mathcal{E}_2)$. Clearly $\mathbb{P}(\mathcal{E}_1) \leq (df_r)^h$. By Lemma 5.14, if all the cells in row j have failed, then at least $\frac{d+1}{2}$ of the elements in row j have failed, so $\mathbb{P}(\mathcal{E}_2) \leq hf_d$. ∎

LEMMA 5.17. *For every $\delta < \frac{1}{2}$ there exists $\varepsilon > 0$ such that, when $0 \leq p \leq \frac{1}{2} - \delta$ and $r \geq \frac{2}{\varepsilon} \ln d$, then*

$$F_p(\text{SC-Grid}) \leq d^{-h} + he^{-\varepsilon d}.$$

Proof. By a Chernoff inequality, there exists $\varepsilon > 0$ such that $f_x \leq e^{-\varepsilon x}$ for all x when $0 \leq p \leq \frac{1}{2} - \delta$. For this ε, if $r \geq \frac{2}{\varepsilon} \ln d$ then $f_r \leq 1/d^2$. Plugging this into Lemma 5.16 finishes the lemma. ∎

By plugging the parameter values into Lemmas 5.15 and 5.17 we obtain the following result.

PROPOSITION 5.18. *For every $\delta < \frac{1}{2}$ there exists $\varepsilon > 0$ such that if $0 \leq p \leq \frac{1}{2} - \delta$, then taking $r = \lceil \frac{2}{\varepsilon} \ln d \rceil$, $d = \sqrt{n \ln n}$, and $h = n/d$ gives $F_p(\text{SC-Grid}) = \exp(-\Omega(\sqrt{n \ln n}))$ and $\mathcal{L}(\text{SC-Grid}) = O(\sqrt{(\ln n)}/n)$.*

Remark. The parameters were chosen to minimize the failure probability. The tradeoff between the load and failure probability is tight for this construction.

5.4. The AndOr system. Consider a complete rooted binary tree of height h, rooted at *root*, and identify the $n = 2^h$ leaves of the tree with the system elements. We define two recursive procedures that operate on a subtree rooted at v and return a set of elements.

(i) For a leaf v, $ANDset(v) = ORset(v) = \{v\}$.
(ii) $ANDset(v) = ORset(v.left) \cup ORset(v.right)$.
(iii) $ORset(v)$ has a choice; it can be either $ANDset(v.left)$ or $ANDset(v.right)$.

A quorum in the AndOr system is any set $Q = S \cup R$ where S is an $ANDset(root)$ and R is an $ORset(root)$.

It is easy to think of the AndOr system as the conjunction of two boolean functions corresponding to the top level activations of *ANDset* and *ORset*. Each function is defined by a complete tree of alternating AND and OR gates, over the same inputs, but one function has an AND gate at the root while the other has an OR gate at the root.

LEMMA 5.19. *If* $S = ANDset(root)$ *and* $R = ORset(root)$, *then* $|S \cap R| = 1$ *for any choices made by the activations of the ORset procedure. Hence* AndOr *is a quorum system.*

Proof. The proof is by induction on the tree height h. The case $h = 0$ is obvious. For $h \geq 1$, assume w.l.o.g. that the *ORset* procedure uses the left subtree. Then any element in the right subtree is not in the intersection, and by the induction hypothesis the intersection in the left subtree has size 1. □

LEMMA 5.20. *The* AndOr *system is a fair system, with*

$$c(\text{AndOr}) = \begin{cases} 2\sqrt{n} - 1, & h \text{ even,} \\ 3\sqrt{n/2} - 1, & h \text{ odd.} \end{cases}$$

Proof. The fairness is obvious from symmetry. Let $ANDsize(h) = |ANDset(root)|$ denote the size of the output of the *ANDset* procedure on a tree with height h, and similarly let $ORsize(h) = |ORset(root)|$. Then by definition, $ANDsize(0) = ORsize(0) = 1$, and

$$\begin{aligned} ANDsize(h) &= 2ORsize(h-1), \\ ORsize(h) &= ANDsize(h-1). \end{aligned}$$

It is easy to show by induction on h that $ORsize(h) = 2^{\lfloor \frac{h}{2} \rfloor}$ and $ANDsize(h) = 2^{\lfloor \frac{h+1}{2} \rfloor}$. Combining with Lemma 5.19 finishes the proof. □

PROPOSITION 5.21. $\mathcal{L}(\text{AndOr}) = O(1/\sqrt{n})$.

Proof. To obtain the proof, apply Proposition 4.10 using Lemma 5.20. □

The following proposition shows the high availability of the AndOr system. The proof is an adaptation of the proof in [43]. We include it here for completeness, omitting some of the technical details.

PROPOSITION 5.22. *Let* $\alpha = \frac{3-\sqrt{5}}{2} \approx 0.38$. $F_p(\text{AndOr}) \leq \exp(-\Omega(\sqrt{n}))$ *when* $p < \frac{1}{4}$ *and* $F_p \leq \exp(-\Omega(n^{0.19}))$ *when* $p \leq \alpha - \Omega(n^{-0.19})$.

Proof. Let $f_A(h)$ denote the probability that all of the possible outputs sets of the *ANDset* procedure are hit, and similarly let f_O denote the probability for the *ORset* procedure, on a tree with height h. Clearly $F_p(\text{AndOr}) \leq f_A(h) + f_O(h)$. By the definitions,

$$\begin{aligned} f_A(h) &= 2f_A^2(h-2) - f_A^4(h-2), \\ f_O(h) &= 4f_O^2(h-2) - 4f_O^3(h-2) + f_O^4(h-2), \end{aligned}$$

and $f_A(0) = f_O(0) = p$. Obviously $f_A(h) < 2f_A^2(h-2)$, and also $f_O(h) = f_O^2(h-2)$ $(2 - f_O(h-2))^2 < 4f_O^2(h-2)$. Therefore, by induction, when h is even,

$$f_A(h) < 2^{2^{\frac{h}{2}}-1} p^{2^{\frac{h}{2}}} < (2p)^{\sqrt{n}}$$

and similarly $f_O(h) < (4p)^{\sqrt{n}}$. So it follows that $F_p(\text{AndOr}) \leq \exp(-\Omega(\sqrt{n}))$ when $p < \frac{1}{4}$. When h is odd the bound is similar.

Now f_O has a stable point at $p = \alpha$ and f_A has a stable point at $p = 1 - \alpha$. As shown by [43], if there are $n = O(d^{5.3})$ leaves in the tree and $p < \alpha - \Omega(d^{-1}) < 1 - \alpha - \Omega(d^{-1})$, then $f_A(h) < 2^{-d-1}$, and the same is true for f_O. Setting $d = O(n^{1/5.3}) = O(n^{0.19})$ finishes the claim. □

We now describe how to use the AndOr system when some elements have failed. We show an algorithm that finds a nearly optimal strategy w for any given configuration $\mathbf{x}$; the load induced by w is at most twice the optimal load under configuration $\mathbf{x}$, $\mathcal{L}(\text{AndOr}_{\mathbf{x}})$. The description is of an activation at the top level of *ANDset*(*root*), say. The description of the *ORset* activation is identical.

The algorithm is a preprocessing step which needs to be done after each configuration change. It begins by recursively marking the internal nodes in the tree as "alive" or "dead" in the obvious way; an AND node is alive if both of its children are alive, and an OR node is alive if at least one of its children is alive.

Consider a live node v. If it is either an AND node, or an OR node with a single live child, then any strategy that chooses to use (elements in the tree rooted at) v is forced to use all of v's live children. Therefore, to complete the description of our strategy w we need to show what happens at OR nodes with two live children. For this, during the preprocessing each such node v is given a probability $\beta(v)$. If the strategy w decides to use v's tree, then it uses its left subtree with probability $\beta(v)$ and its right subtree with probability $1 - \beta(v)$.

To compute the $\beta(v)$ values for live OR nodes v with two live children, the algorithm recursively computes the optimal loads ℓ_L and ℓ_R in the left and right subtrees, respectively. To achieve an optimal load for v's tree, $\beta(v)$ must satisfy $\beta\ell_L = (1-\beta)\ell_R$. Therefore, $\beta(v) = \ell_R/(\ell_L + \ell_R)$, and the load induced on v's tree is $\ell_L\ell_R/(\ell_L + \ell_R)$.

The above computation is performed twice, once starting with *ANDset*(*root*) and once starting with *ORset*(*root*). Note that a node may be marked "alive" w.r.t. the *ANDset*(*root*) activation and "dead" w.r.t. the *ORset*, or vice versa. However, every v is assigned a single $\beta(v)$ value since it is an OR node only w.r.t. one top level activation.

This w would clearly induce an optimal load for any configuration $\mathbf{x}$ if we were interested in a single top-level activation. However, since we must activate both *ANDset* and *ORset* at the top level, a moment's reflection shows that $\mathcal{L}_w(\text{AndOr}_{\mathbf{x}}) \leq 2\mathcal{L}(\text{AndOr}_{\mathbf{x}})$.

Remarks.

(i) A quorum system can be constructed from any monotone read-once boolean function in a similar way. This is achieved by taking some AND/OR formula F implementing the function and making a dual copy of it F^d (in which every AND gate is replaced by an OR gate and vice versa). A quorum is defined to be a union of two sets of elements, one satisfying F and the other satisfying F^d. The proof of Lemma 5.19 would still hold for such a system. However, the load and failure probability would depend on the specific structure of the function used.

(ii) The AndOr system is isomorphic to the hierarchical grid construction of [24], when the grids at all the levels are 2×2 grids. The read-quorum and write-quorum procedures of [24] correspond to our top-level activations of the *ANDset* and *ORset* procedures, respectively. However, ours is a much stronger analysis; we calculate the load and analyze the rate of decay of F_p and the critical probability α.

6. Load analyses of some quorum systems.

6.1. Nondominated coteries have lower loads. The following proposition shows that nondominated coteries (see Definition 2.5) have the lowest loads. This

gives further support to the intuitive view that NDC's are preferable to dominated coteries for practical applications.

PROPOSITION 6.1. *Let $\mathcal{S}$, $\mathcal{R}$ be quorum systems over the same universe U such that $\mathcal{R} \succ \mathcal{S}$. Then $\mathcal{L}(\mathcal{R}) \leq \mathcal{L}(\mathcal{S})$.*

Proof. Assume that $\mathcal{S} = \{S_1, \ldots, S_m\}$ and $\mathcal{R} = \{R_1, \ldots, R_{m'}\}$. Define a mapping $\varphi : \mathcal{S} \mapsto \mathcal{R}$ as follows. For every set $S_k \in \mathcal{S}$ consider the minimal j such that $R_j \subseteq S_k$, and let $\varphi(S_k) = R_j$. By Definition 2.5 there exists such an R_j for every S_k, so φ is well defined. Let w be an optimal strategy for $\mathcal{S}$. Define w' for $\mathcal{R}$ by

$$w'_j = \begin{cases} \sum_{\varphi(S_k)=R_j} w_k, & \text{if } \exists k : \varphi(S_k) = R_j, \\ 0, & \text{otherwise.} \end{cases}$$

Clearly w' is a strategy for $\mathcal{R}$. The load induced by strategy w' on an element i is

$$\ell_{w'}(i) = \sum_{R_j \ni i} w'_j = \sum_{R_j \ni i} \left(\sum_{\varphi(S_k)=R_j} w_k \right) \leq \sum_{S_k \ni i} w_k = \ell_w(i).$$

Applied to the load on the busiest element i this implies that

$$\mathcal{L}_{w'}(\mathcal{R}) \leq \mathcal{L}_w(\mathcal{S}) = \mathcal{L}(\mathcal{S}),$$

and by the minimality of $\mathcal{L}(\mathcal{R})$ the result follows. □

Remark. Proposition 6.1 does not imply that *dominated* quorum systems necessarily have a *high* load. In fact, all our constructions in section 5 are dominated and have optimal or near-optimal load. By Proposition 6.1 there exist NDC's with loads which are as good or better—but these are more cumbersome to describe explicitly.

6.2. Voting systems have high loads. A popular and simple way to construct a quorum system is by *weighted voting* [14, 13, 41, 29]. In this section we show that $\mathcal{L}(\mathcal{S}) > \frac{1}{2}$ for any voting system $\mathcal{S}$, i.e., any voting system is at least as bad as the Maj system in terms of load.

DEFINITION 6.2. *For each $i \in U$ let the integer $v_i \geq 0$ denote the* weight *of i. Let $V = \sum_i v_i$ be the* total weight*. The* voting system *defined by the weights v_i is*

$$\text{Vote} = \left\{ S \subseteq U : \sum_{i \in S} v_i > \frac{V}{2} \right\}.$$

PROPOSITION 6.3. $\mathcal{L}(\text{Vote}) > \frac{1}{2}$.

Proof. Consider the vector $\mathbf{y}$ defined by $y_i = v_i/V$ for all $i \in U$. Clearly $\mathbf{y}(U) = 1$. By Definition 6.2,

$$\mathbf{y}(S) = \frac{1}{V} \sum_{i \in S} v_i > \frac{1}{2},$$

for any quorum $S \in \text{Vote}$. Therefore $(\mathbf{y}; \frac{1}{2})$ is a feasible point to program DLP, so $\mathcal{L}(\text{Vote}) > \frac{1}{2}$ by the weak duality of linear programming. □

6.3. The tree system. We have shown in Example 4.4 that the load of the Tree quorum system [1] is $\mathcal{L}(\text{Tree}) \geq \frac{1}{\log(n+1)}$. In this section we show that the bound is almost tight; the precise load is $\mathcal{L}(\text{Tree}) = \frac{2}{\log(n+1)+1}$. We first show an upper bound

by balancing the load on the elements, and then show a matching lower bound. We use h to denote the height of the tree ($n = 2^{h+1} - 1$).

CLAIM 6.4. $\mathcal{L}(\text{Tree}) \leq \frac{2}{h+2}$.

Proof. Denote a tree rooted at node i by $T(i)$, and denote its left and right subtrees by $T_L(i)$ and $T_R(i)$. We build a probabilistic recursive strategy *Pick* to pick a quorum, using values β_h, to be defined later, as follows.

$$Pick(T(i)) = \begin{cases} \{i\} \cup Pick(T_L(i)), & \text{with probability } \beta_h, \\ \{i\} \cup Pick(T_R(i)), & \text{with probability } \beta_h, \\ Pick(T_L(i)) \cup Pick(T_R(i)), & \text{with probability } 1 - 2\beta_h. \end{cases}$$

Let $L(h)$ denote the load induced by strategy *Pick* in a tree of height h. The load is determined either by the load on the root i, or by the most heavily loaded element in one of the subtrees. Therefore $L(h) = \max\{2\beta_h, (1 - \beta_h)L(h-1)\}$. Choosing $\beta_h = \frac{L(h-1)}{L(h-1)+2}$ balances the load, so with this choice the load obeys the recurrence

$$L(h) = \frac{2L(h-1)}{L(h-1)+2},$$

and $L(0) = 1$. A simple check shows that $L(h) = \frac{2}{h+2}$ solves this recurrence, and then $\beta_h = \frac{1}{h+2}$ for $h \geq 1$. □

CLAIM 6.5. $\mathcal{L}(\text{Tree}) \geq \frac{2}{h+2}$.

Proof. Let $0 \leq t_i \leq h$ denote the distance from node i to the root. To show a matching lower bound we build a dual-feasible vector of weights $\mathbf{y}$, defined by

$$y_i = \begin{cases} \frac{1}{h+2}\left(\frac{1}{2}\right)^{t_i}, & 0 \leq t_i < h, \\ \frac{1}{h+2}\left(\frac{1}{2}\right)^{h-1}, & t_i = h. \end{cases}$$

It is easy to see that $\mathbf{y}$ is a valid weight vector. We need to show that $\mathbf{y}(S) \geq \frac{2}{h+2}$ for every quorum $S \in \text{Tree}$.

By induction from the leaves toward the root, one can show that

$$\mathbf{y}(S \cap T(i)) = \begin{cases} \frac{2}{h+2}\left(\frac{1}{2}\right)^{t_i}, & S \cap T(i) \neq \varnothing, \\ 0, & \text{otherwise}, \end{cases} \tag{5}$$

for every $i \in U$ and $S \in \text{Tree}$. Plugging the root of the tree we obtain $\mathbf{y}(S) = \frac{2}{h+2}\left(\frac{1}{2}\right)^0 = \frac{2}{h+2}$. Therefore, $(\mathbf{y}; \frac{2}{h+2})$ is feasible for program DLP so the claim follows from the weak duality of linear programming. □

6.4. The hierarchical quorum system. In this section we analyze the load and availability of the hierarchical system of [23]. In this system the elements are the leaves of a complete ternary tree. The internal nodes are 2-of-3 majority gates. We show that $F_p(\text{HQS}) \leq \exp(-\Omega(n^{0.63}))$ when $p < \frac{1}{3}$ and $F_p(\text{HQS}) \leq n^{-\alpha(p)}$ when $p < \frac{1}{2}$, and that $\mathcal{L}(\text{HQS}) = n^{-0.37}$.

The analysis is similar in nature to that of the AndOr system. However, HQS is a nondominated system, so the analysis is good up to $\frac{1}{2}$ rather than up to the 0.38 of the AndOr system. On the other hand, the load of HQS is worse than the $O(1/\sqrt{n})$ of the AndOr system.

We use h to denote the height of the tree ($n = 3^h$).

PROPOSITION 6.6. $\mathcal{L}(\text{HQS}) = n^{-0.37}$.

Proof. By symmetry it follows that HQS is a fair system, with $c(\text{HQS}) = n^{\log_3 2} = n^{0.63}$. Therefore by Proposition 4.10, $\mathcal{L}(\text{HQS}) = n^{0.63}/n = n^{-0.37}$. □

PROPOSITION 6.7. *$F_p(\text{HQS}) \leq \exp(-\Omega(n^{0.63}))$ when $p < \frac{1}{3}$ and $F_p(\text{HQS}) \leq n^{-\alpha(p)}$ when $p < \frac{1}{2}$.*

Proof. Let $f(h)$ denote $F_p(\text{HQS})$ on a tree with height h. Then $f(h)$ obeys the recurrence

$$f(h) = 3f^2(h-1) - 2f^3(h-1),$$

and $f(0) = p$. We observe that $p = \frac{1}{2}$ is a stable point, so by a result of [35] it follows that HQS is nondominated. Now certainly $f(h) < 3f^2(h-1)$, so by induction on h we show that $f(h) < 3^{2^h-1}p^{2^h} < (3p)^{n^{\log_3 2}}$, which proves the case when $p < \frac{1}{3}$.

For larger values of p, we prove by induction on h that $f(h)$ is decreasing when $p < \frac{1}{2}$, and then, that

$$f(h) \leq p \cdot (3p - 2p^2)^h.$$

If $p = \frac{1}{2} - \varepsilon$ then $3p - 2p^2 < 1 - \varepsilon$ so $f(h) < \frac{1}{2}(1-\varepsilon)^{\log_3 n} < n^{-\alpha}$ for some $\alpha(p)$. □

Remark. The HQS system has a tight tradeoff between its availability and load when $p < \frac{1}{3}$.

Appendix. Results of percolation theory. In this section we list the definitions and results that are used in our analysis of the Paths system, following [15].

The percolation model we are interested in is as follows. Let $\mathbb{Z}^2$ be the graph of the square lattice in the plane. Assume that an edge between neighboring vertices in $\mathbb{Z}^2$ is *closed* with probability p and *open* with probability $q = 1 - p$, independently of other edges. This model is known as bond percolation on the square lattice. Another natural model, which plays a minor role in our work, is the site percolation model. In it the *vertices* are closed with probability p. Unless otherwise stated, we always use the bond percolation model.

Notation. For an event $\mathcal{E}$ defined in the percolation model (either on $\mathbb{Z}^2$ or on some finite subgraph of $\mathbb{Z}^2$), we denote the probability of $\mathcal{E}$ by $\mathbb{P}_p(\mathcal{E})$.

A key idea in percolation theory is that there exists a *critical probability*, p_c, such that graphs with $p < p_c$ exhibit qualitatively different properties than graphs with $p > p_c$. For example, $\mathbb{Z}^2$ with $p < p_c$ has a single connected (open) component of infinite size. When $p > p_c$ there is no such component (see [15, p. 110]). For bond percolation in the plane $p_c = \frac{1}{2}$ [22].

Notation. Let $B(d)$ be the ball of radius d with center at the origin; $B(d) = \{v \in \mathbb{Z}^2 : |v_1| + |v_2| \leq d\}$. Let $\partial B(d)$ be the surface of $B(d)$, $\partial B(d) = \{v \in \mathbb{Z}^2 : |v_1| + |v_2| = d\}$. For a vertex v and a set of vertices A, let $v \leftrightarrow A$ denote the event that there exists an open path between v and some vertex in A.

The following theorem shows that when the probability p for a closed edge is above the critical probability, the probability of having long open paths decays exponentially fast.

THEOREM A.1. (See [31].) *If $p > \frac{1}{2}$, then there exists $\psi(p) > 0$ such that*

$$\mathbb{P}_p(0 \leftrightarrow \partial B(d)) < e^{-\psi(p)d} \quad \textit{for all } d.$$

DEFINITION A.2. *Let $\mathcal{E}$ be an event defined in the percolation model. Then the* interior *of $\mathcal{E}$ with depth r, denoted $I_r(\mathcal{E})$, is the set of all configurations in $\mathcal{E}$ which are still in $\mathcal{E}$ even if we perturb the states of up to r edges.*

We may think of $I_r(\mathcal{E})$ as the event that $\mathcal{E}$ occurs and is "stable" with respect to changes in the states of r or fewer edges. The definition is useful to us in the following situation. If LR is the event "there exists an open left-right path in a rectangle D," then by flow considerations it follows that $I_r(LR)$ is the event "there are at least $r+1$ edge disjoint open left-right paths in D."

THEOREM A.3. (See [3].) *Let $\mathcal{E}$ be an increasing event and let r be a positive integer. Then*

$$1 - \mathbb{P}_p(I_r(\mathcal{E})) \leq \left(\frac{q}{q-q'}\right)^r [1 - \mathbb{P}_{p'}(\mathcal{E})]$$

whenever $0 \leq p < p' \leq 1$.

The theorem amounts to the assertion that if $\mathcal{E}$ is likely to occur when the edge failure probability is p', then $I_r(\mathcal{E})$ is likely to occur when the failure probability is smaller than p'.

Acknowledgments. We are grateful to David Peleg for his encouragement and to Danny Raz for his careful reading of our manuscript. We thank the anonymous referees for remarks which helped us improve our presentation, and for bringing [12] and [45] to our attention.

REFERENCES

[1] D. AGRAWAL AND A. EL-ABBADI, *An efficient and fault-tolerant solution for distributed mutual exclusion*, ACM Trans. Comp. Sys., 9 (1991), pp. 1–20.

[2] R. AHARONI, P. ERDŐS, AND N. LINIAL, *Dual integer linear programs and the relationship between their optima*, in Proc. 17th Annual ACM Symp. Theory of Computing (STOC), ACM, New York, 1985, pp. 476–483.

[3] M. AIZENMAN, J. T. CHAYES, L. CHAYES, J. FRÖHLICH, AND L. RUSSO, *On a sharp transition from area law to perimeter law in a system of random surfaces*, Comm. Math. Physics, 92 (1983), pp. 19–69.

[4] D. BARBARA AND H. GARCIA-MOLINA, *The reliability of vote mechanisms*, IEEE Trans. Comput., C-36 (1987), pp. 1197–1208.

[5] S. Y. CHEUNG, M. H. AMMAR, AND M. AHAMAD, *The grid protocol: A high performance scheme for maintaining replicated data*, in Proc. 6th IEEE Int. Conf. Data Engineering, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 438–445.

[6] N. CONDORCET, *Essai sur l'application de l'analyse à la probabilité des decisions rendues à la pluralite des voix*, Paris, 1785.

[7] S. B. DAVIDSON, H. GARCIA-MOLINA, AND D. SKEEN, *Consistency in partitioned networks*, ACM Computing Surveys, 17 (1985), pp. 341–370.

[8] M. DUBINER AND U. ZWICK, *Amplification and percolation*, in Proc. 33rd Annual IEEE Symp. Foundations of Comput. Sci. (FOCS), IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 258–267.

[9] P. ERDŐS AND L. LOVÁSZ, *Problems and results on 3-chromatic hypergraphs and some related questions*, in Infinite and Finite Sets, Proc. Colloq. Math. Soc. János Bolyai 10, 1975, North–Holland, Amsterdam, pp. 609–627.

[10] C. M. FORTUIN, P. W. KASTELEYN, AND J. GINIBRE, *Correlation inequalities on some partially ordered sets*, Comm. Math. Physics, 22 (1971), pp. 89–103.

[11] G. N. FREDERICKSON, *Fast algorithms for shortest paths in planar graphs, with applications*, SIAM J. Comput., 16 (1987), pp. 1004–1022.

[12] Z. FÜREDI, *Matchings and covers in hypergraphs*, Graphs Combin., 4 (1988), pp. 115–206.

[13] H. GARCIA-MOLINA AND D. BARBARA, *How to assign votes in a distributed system*, J. ACM, 32 (1985), pp. 841–860.

[14] D. K. GIFFORD, *Weighted voting for replicated data*, in Proc. 7th Annual ACM Symp. Oper. Sys. Principles (SIGOPS), ACM, New York, 1979, pp. 150–159.

[15] G. R. GRIMMETT, *Percolation*, Springer-Verlag, New York, 1989.

[16] M. GRÖTSCHEL, L. LOVÁSZ, AND A. SCHRIJVER, *The ellipsoid method and its consequences in combinatorial optimization*, Combinatorica, 1 (1981), pp. 169–197.

[17] R. HASSIN, *Maximum flow in (s,t) planar networks*, Inform. Process. Lett., 13 (1981), p. 107.

[18] M. P. Herlihy, *Replication Methods for Abstract Data Types*, Ph.D. thesis MIT/LCS/TR-319, Massachusetts Institute of Technology, Cambridge, MA, 1984.
[19] R. Holzman, Y. Marcus, and D. Peleg, *Load balancing in quorum systems*, SIAM J. Discrete Math., 10 (1997), pp. 223–245.
[20] A. Itai and Y. Shiloach, *Maximum flow in planar networks*, SIAM J. Comput., 8 (1979), pp. 135–150.
[21] P. Jalote, S. Rangarajan, and S. K. Tripathi, *Capacity of voting systems*, IEEE Trans. Software Eng., 19 (1993), pp. 698–706.
[22] H. Kesten, *The critical probability of bond percolation on the square lattice equals* $\frac{1}{2}$, Comm. Math. Physics, 71 (1980), pp. 41–59.
[23] A. Kumar, *Hierarchical quorum consensus: A new algorithm for managing replicated data*, IEEE Trans. Comput., 40 (1991), pp. 996–1004.
[24] A. Kumar and S. Y. Cheung, *A high availability* $\sqrt{n}$ *hierarchical grid algorithm for replicated data*, Inform. Process. Lett., 40 (1991), pp. 311–316.
[25] A. Kumar, M. Rabinovich, and R. K. Sinha, *A performance study of general grid structures for replicated data*, in Proc. 13th Int. Conf. Dist. Comp. Sys., IEEE Computer Society Press, Los Alamitos, CA, 1993, pp. 178–185.
[26] L. Lovász, *Coverings and colorings of hypergraphs*, in Proc. 4th Southeastern Conf. Combinatorics, Graph Theory and Computing, 1973, Utilitas Math. Publishing Inc., Winnipeg, pp. 3–12.
[27] L. Lovász, *An Algorithmic Theory of Numbers, Graphs and Convexity*, SIAM, Philadelphia, 1986.
[28] M. Maekawa, *A* $\sqrt{n}$ *algorithm for mutual exclusion in decentralized systems*, ACM Trans. Comp. Sys., 3 (1985), pp. 145–159.
[29] Y. Marcus and D. Peleg, *Construction Methods for Quorum Systems*, Tech. Report CS92–33, The Weizmann Institute of Science, Rehovot, Israel, 1992.
[30] T. R. Mathies, *Percolation theory and computing with faulty arrays of processors*, in Proc. 3rd ACM-SIAM Symp. on Discrete Alg., SIAM, Philadelphia, 1992, pp. 100–103.
[31] M. V. Menshikov, *Coincidence of critical points in percolation problems*, Soviet Math. Dok., 33 (1986), pp. 856–859.
[32] S. J. Mullender and P. M. B. Vitányi, *Distributed match-making*, Algorithmica, 3 (1988), pp. 367–391.
[33] M. Naor and A. Wool, *The load, capacity and availability of quorum systems*, in Proc. 35th Annual IEEE Symp. Foundations of Comput. Sci. (FOCS), IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 214–225.
[34] M. Naor and A. Wool, *Access control and signatures via quorum secret sharing*, in Proc. 3rd ACM Conf. Comp. and Comm. Security, New Delhi, India, ACM, New York, 1996, pp. 157–168. Also available as Theory of Cryptography Library record 96-08, http://theory.lcs.mit.edu/∼tcryptol/1996.html.
[35] D. Peleg and A. Wool, *The availability of quorum systems*, Inform. and Comput., 123 (1995), pp. 210–223.
[36] D. Peleg and A. Wool, *The availability of crumbling wall quorum systems*, Discrete Appl. Math., 74 (1997), pp. 69–83.
[37] D. Peleg and A. Wool, *Crumbling walls: A class of practical and efficient quorum systems*, Distrib. Comput., 10 (1997), pp. 87–98.
[38] S. Rangarajan and S. K. Tripathi, *A robust distributed mutual exclusion algorithm*, in Proc. 5th Int. Workshop on Dist. Algorithms (WDAG), Lecture Notes in Comput. Sci. 579, Springer-Verlag, New York, 1991, pp. 295–308.
[39] M. Raynal, *Algorithms for Mutual Exclusion*, MIT Press, Cambridge, MA, 1986.
[40] A. Schrijver, *Theory of Linear and Integer Programming*, John Wiley, Chichester, 1986.
[41] M. Spasojevic and P. Berman, *Voting as the optimal static pessimistic scheme for managing replicated data*, IEEE Trans. Par. Dist. Sys., 5 (1994), pp. 64–73.
[42] R. H. Thomas, *A majority consensus approach to concurrency control for multiple copy databases*, ACM Trans. Database Sys., 4 (1979), pp. 180–209.
[43] L. G. Valiant, *Short monotone formulae for the majority function*, J. Algorithms, 5 (1984), pp. 363–366.
[44] C. Wu, *Replica Control Protocols that Guarantee High Availability and Low Access Cost*, Tech. Report 1817, Dept. Computer Science, University of Illinois, Urbana-Champaign, Urbana, Illinois, 1993.
[45] C. Wu and G. G. Belford, *The triangular lattice protocol: A highly fault tolerant protocol for replicated data*, in Proc. 11th Annual IEEE Symp. on Reliable Dist. Sys., 1992, pp. 66–73.
[46] T. W. Yan and H. Garcia-Molina, *Distributed selective dissemination of information*, in Proc. 3rd Int. Conf. Par. Dist. Info. Sys., IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 89–98.

SIAM J. COMPUT.
Vol. 27, No. 2, pp. 448–465, April 1998
© 1998 Society for Industrial and Applied Mathematics
007

SPACE-EFFICIENT DETERMINISTIC SIMULATION OF PROBABILISTIC AUTOMATA*

IOAN I. MACARIE†

Abstract. Given a description of a probabilistic automaton (one-head probabilistic finite-state automaton or probabilistic Turing machine) and an input string x of length n, we ask how much space does a deterministic Turing machine need in order to decide the acceptance of the input string by that automaton?

The question is interesting even in the case of one-head one-way probabilistic finite-state automata (1pfa's). We call (*rational*) *stochastic languages* ($S^{>}_{rat}$) the class of languages recognized by 1pfa's whose transition probabilities and cutpoints (i.e., recognition thresholds) are rational numbers. The class $S^{>}_{rat}$ contains context-sensitive languages that are not context-free, but on the other hand there are context-free languages not included in $S^{>}_{rat}$.

Our main results are as follows:

- The (proper) inclusion of $S^{>}_{rat}$ in Dspace($\log n$), which is optimal (i.e., $S^{>}_{rat} \not\subset$ Dspace($o(\log n)$)). The previous upper bounds were Dspace(n) (obtained by P. D. Dieu in 1972) and Dspace($\log n \log\log n$) (obtained by H. Jung in 1984).
- Probabilistic Turing machines with space bound $f(n) \in O(\log n)$ can be deterministically simulated in space $O(\min(c^{f(n)} \log n, \log n(f(n)+\log\log n)))$, where c is a constant depending on the simulated probabilistic Turing machine. The best previously known simulation uses $O(\log n(f(n) + \log\log n))$ space (obtained by H. Jung in 1984).

To obtain these results we develop and use two space-efficient (but also parallel-time-efficient) techniques, which are of independent interest:

- a technique to compare deterministically, using only $O(\log n)$ space, large numbers given in terms of their values modulo a sequence of primes, $p_1 < p_2 < \cdots < p_n \in O(n^a)$ (where a is some arbitrary constant);
- a technique to compare deterministically a threshold with entries of the inverse of a given banded matrix.

Key words. stochastic languages, probabilistic Turing machines, deterministic simulation, residue representation, space-bounded complexity classes, matrix inversion

AMS subject classifications. 68Q15, 68Q75, 68Q05, 68Q10, 68Q22, 68R99

PII. S0097539793253851

1. Introduction.

The *(one-head one-way) probabilistic finite(-state) automaton* (1pfa) was introduced by Rabin in 1963 as a generalization of the deterministic finite automaton. He proved that, in general, probabilistic automata are stronger than their deterministic counterparts. The concepts of *cutpoint* and *isolated cutpoint* are due to the same author. He also showed that probabilistic automata with isolated cutpoint[1] recognize only regular languages, and he stated a relation between the complexities (i.e., the minimal numbers of states) of probabilistic automata and the complexities of deterministic automata that recognize the same languages. The languages recognized by one-head one-way probabilistic finite automata are called

*Received by the editors August 9, 1993; accepted for publication (in revised form) February 8, 1996. An extended abstract of this paper appears in *Proceedings of the 11th Symposium on Theoretical Aspects of Computer Science*, 1994, Lecture Notes in Comput. Sci. 775, Springer-Verlag, New York, pp. 109–122.

http://www.siam.org/journals/sicomp/27-2/25385.html

†Millennium Computer Corporation, Research and Advanced Development Group, 2851 Clover Street, Pittsford, NY 14534 (macarie@millenniumcc.com). This research was mainly supported by the National Science Foundation under grant CDA 8822724 while the author was working in the Department of Computer Science, University of Rochester, Rochester, NY.

[1]Probabilistic automata with isolated cutpoint are also known as probabilistic automata that recognize with bounded-error.

stochastic languages ($S^{>}$). Turakainen [Tu68] defined the concepts of *generalized probabilistic finite automaton* and *generalized stochastic language* and proved the equality between the class of stochastic languages and the class of generalized stochastic languages. Dieu [Di72] showed that *rational stochastic languages* ($S^{>}_{rat}$), i.e., languages recognized by probabilistic finite automata with rational transition probabilities and rational cutpoint, can be recognized by $O(n)$-space-bounded deterministic Turing machines.

In 1973, Kuklin [Ku73] introduced the *(one-head) two-way probabilistic finite automaton* (2pfa) as a generalization of 1pfa. Some time later, Freivalds [Fr81] proved that the language $\{a^n b^n \mid n \in \mathbf{N}\}$ can be recognized, although slowly, by a *two-way probabilistic finite automaton with isolated cutpoint* (2bpfa). Jung [Ju84] found an $O(\log n \log\log n)$-space deterministic simulation for 2pfa's, and later Wang [Wa92] provided a much weaker $O(n)$-space deterministic simulation for 2bpfa's using an interesting technique based on the Markov chain tree theorem of Leighton and Rivest [LR86]. In spite of the fact that the classes of languages recognized by 1pfa's and 2pfa's are the same (if their transition probabilities are either all rationals or all reals) [Ka89], the class of languages recognized by 2bpfa's is larger than the class of languages recognized by 1pfa's with isolated cutpoint. Jung [Ju84] obtained an optimal $O(\log n)$ deterministic space simulation for 2bpfa's.

The significance of finite-state automata is limited by their weakness. Therefore, in the 1970's the general interest moved to more powerful devices. Gill [Gi77] introduced the concept of *probabilistic Turing machines* (PTM). Later Simon [Si77, Si81], Borodin, Cook, and Pippenger [BCP83], and Jung [Ju81, Ju84] obtained several results on space-bounded (unbounded-error) probabilistic computation. Probably the most important result is that the languages recognized by PTM's in space $f(n) \in \Omega(\log n)$ can be recognized by deterministic Turing machines in space $O(f^2(n))$. This result, together with the fact that $f(n)$-space PTM's are at least as strong as $f(n)$-space nondeterministic Turing machines, generalizes Savitch's theorem [Sa70]. In the case of small space-bounded probabilistic complexity classes, Jung proved the inclusion of languages recognized by $f(n) \in O(\log n)$ space-bounded PTM's in $\mathrm{Dspace}(\log n(f(n) + \log\log n))$ [Ju84].

In this article we present space-efficient simulations of small space-bounded (unbounded-error) PTM's by derandomized (i.e., deterministic) Turing machines: we show a more efficient simulation of $f(n) \in o(\log n)$ space-bounded PTM's and an optimal $O(\log n)$-space simulation of one-head probabilistic finite automata with rational transition probabilities. Our simulations are presented for the more general setting of Markov chains, so they may have other applications as well. To obtain these results we develop and use two techniques that are of independent interest:

- a technique to compare deterministically, using only $O(\log n)$ space, numbers given in terms of their values modulo a sequence of primes, $p_1 < p_2 < \cdots < p_n \in O(n^a)$ (where a is some arbitrary constant);
- a technique to compare deterministically a threshold with entries of the inverse of a given banded matrix.

In section 2 we define the notions used in this article and we recall some basic results on stochastic languages.

Section 3 presents optimal deterministic logspace simulations for one-head probabilistic finite automata (Theorems 3–4), and a space-efficient technique to work with residue representations (Lemmas 1–3).

In section 4 we show that probabilistic Turing machines with space-bound $f(n) \in O(\log n)$ can be simulated by deterministic Turing machines that use only

$O(\min(c^{f(n)} \log n, \log n(\log f(n) + \log\log n)))$ space, where c is a constant depending on the simulated machine (Theorem 5).

Finally, in section 5 we discuss the significance of our results and we state some open problems.

2. Background. This section presents the definitions and the notations used in this article, and gives reminders of some properties of stochastic languages.

DEFINITION 1. (See [Ra63].) *A* (one-head one-way) probabilistic finite automaton (1pfa) *A is a* 5*-tuple*

$$A = (Q, \Sigma, \pi, \{(x, M(x)) | x \in \Sigma\}, F),$$

where

Q is the (finite) set of states, whose cardinal is denoted by n (i.e., $n = |Q|$);

Σ is the input alphabet;

π is the initial state-distribution vector, satisfying $\sum_{i=1}^{n} \pi(i) = 1$, where $\pi(i) \in [0, 1]$;

F is the set of accepting states, $F \subset Q$ (implicitly, $Q - F$ is the set of rejecting states);

$M(x)$ is an n-by-$n stochastic matrix*, whose entry $m_{ij}(x)$ is the probability of transition from state s_i to state s_j under the input symbol x, and where $\sum_{j=1}^{n} m_{ij}(x) = 1$, for each $i \in \{1, \ldots, n\}$.*

DEFINITION 2. (See [Tu68].) *A* generalized (one-head one-way) probabilistic finite automaton (1Gpfa) *is a* 5*-tuple as above with the following relaxed constraints: π is an n-dimensional real vector (and not a state-distribution vector), and the transition matrices $M(x)$ are arbitrary real matrices not constrained to be stochastic. Such a machine is also known as a* weighted finite automaton [CK94].

Observation 1. Equivalently, a 1pfa (1Gpfa) is a finite-state automaton that has one head restricted to move from left to right on an input string $y = x_1, \ldots, x_k$. (We consider that the length of y is k, and that $x_i, i \in \{1, \ldots, k\}$ are symbols from the input alphabet. Also, we may consider that the input string is delimited by a left and a right endmarker.) From a current state i, after reading the symbol x_l scanned by its head the automaton switches to state j with probability (weight) $m_{ij}(x_l)$ and moves its head to the next input symbol.

In what follows, we describe the acceptance procedure of the string y by such an automaton. Let Σ^* be the set of words over the input alphabet Σ, and let ϵ be the empty word. A word matrix $M(y), y \in \Sigma^*$ is the matrix product $M(y) = M(x_1), \ldots, M(x_k)$, where $y = x_1, \ldots, x_k$, $x_i \in \Sigma$, and $M(\epsilon) = I_n$, where I_n is the identity matrix of order n. The state-distribution vector of a probabilistic automaton A after scanning the word $y \in \Sigma^*$ is $r_A(y) = \pi M(y)$. The *acceptance probability* of y by A (i.e., the probability that A reaches an accepting state when processing y) is $p_A(y) = \pi M(y) \eta^F$ where η^F is the column vector whose ith component is equal to 1 or 0 depending on whether or not the ith state is an accepting state.

For a 1Gpfa A, we define the *acceptance function* p_A in a similar way, but in this case p_A is real-valued, and it is not necessarily a probability function anymore.

DEFINITION 3. (See [Ra63].) *The class of* stochastic languages ($S^>$) *is the class containing the languages of the form*

$$T(A, \lambda) = \{y \in \Sigma^* | p_A(y) > \lambda\},$$

where A is a 1pfa *and λ is a real number in* [0,1], *called* cutpoint *(or recognition*

threshold). If $\exists \epsilon > 0$ *such that* $\forall y \in \Sigma^* \Rightarrow |p_A(y) - \lambda| > \epsilon$, *then* λ *is called* isolated cutpoint. *(The recognition with isolated cutpoint is identical to bounded-error recognition.)*

DEFINITION 4. (See [St66].) $\mathrm{S}^{=}$ *is the class of languages of the form* $T^{=}(A, \lambda) = \{y \in \Sigma^* | P_A(y) = \lambda\}$ *where* A *is a* 1pfa *and* λ *is a cutpoint in* [0,1].

DEFINITION 5. $\mathrm{S}^{\neq}$ *is the class of languages of the form* $T^{\neq}(A, \lambda) = \{y \in \Sigma^* | P_A(y) \neq \lambda\}$, *where* A *is a* 1pfa *and* λ *is a cutpoint in* $[0, 1]$.

DEFINITION 6. (See [Tu69b].) *For each class* S^{a} *(* $a \in \{>, =, \neq\}$*) we define the rational subclasses* $\mathrm{S}^{\mathrm{a}}_{\mathrm{rat}}$ *containing the languages of the form* $T^a(A, \lambda)$, *where* A *is a* 1pfa *with the elements of its stochastic matrices and of its initial state-distribution vector all rationals, and* λ *is a rational cutpoint. We have defined in this way* $\mathrm{S}^{>}_{\mathrm{rat}}$, $\mathrm{S}^{=}_{\mathrm{rat}}$, $\mathrm{S}^{\neq}_{\mathrm{rat}}$.

In what follows, we call $\mathrm{S}^{>}_{\mathrm{rat}}$,$\mathrm{S}^{=}_{\mathrm{rat}}$, and $\mathrm{S}^{\neq}_{\mathrm{rat}}$ (rational) stochastic classes. All the probabilistic automata used in this article have rational transition probabilities; for simplicity, sometimes we drop the word "rational" without creating misunderstanding.

It is easy to prove that if in all the previous definitions we fix the value of the threshold λ to be $1/2$, the classes of languages defined so far do not change. Consequently, we will use only probabilistic automata whose cutpoint is $1/2$.

A *one-head two-way probabilistic finite automaton* (2pfa) is similar to a 1pfa, but its head can move in both directions on the input string, which is delimited by a left and right endmarker. Such an automaton has final (accepting and rejecting) states, and it halts when reaching these states. A 2pfa accepts an input string x if its acceptance probability when processing x (i.e., the probability to reach an accepting state) is larger than $1/2$. A one-head two-way probabilistic finite automaton with *isolated cutpoint* (2bpfa) is a 2pfa with the following restriction: there is an open interval I containing the cutpoint such that for any input string x its acceptance probability by the automaton does not fall inside I. A formal definition of these devices can be found in [Ka89]. We use the notations $\mathrm{2PFA}_{rat}$ ($\mathrm{2BPFA}_{rat}$) for the classes of languages recognized by one-head two-way probabilistic finite automata with rational transition probabilities (and isolated cutpoint).

The definition for probabilistic Turing machine (PTM) is standard [Gi77]. To simplify our presentation, in this article we simulate probabilistic Turing machines that have a single head on the input tape and a single head on the worktape. The computation of a space-bounded PTM (or of a probabilistic finite automaton) on an input string is viewed as a Markov process [Gi77, Si77] that is characterized by its state transition matrix. This approach is used in section 4.

Other notations we use in this article are as follows:

- $\subsetneq$ means proper inclusion,
- $\subset$ means inclusion (not necessarily proper),
- $|x|$ is the length of x if x is a string or the absolute value of x if x is a number or the number of elements of x if x is a set,
- $\|X\|$ is the norm of the matrix X defined by $\|X\| = \max_i(\sum_j |x_{ij}|)$,
- $\mathrm{Dspace}(S(n))$, $\mathrm{Nspace}(S(n))$, and $\mathrm{PrSpace}(S(n))$ are the classes of languages recognized by deterministic, nondeterministic, and probabilistic, respectively, $S(n)$-space-bounded Turing machines,
- $\mathbf{N}$ is the set of natural numbers.

Next we recall some closure properties of these classes and we present relations between them and the Chomsky's hierarchy.

PROPOSITION 1. (See [Di71].) *The classes* $S^{=}_{rat}$ *and* $S^{\neq}_{rat}$ *are closed under intersection and union but are not closed under complementation. The complement of any language from one class is in the other class.*

THEOREM 1. (See [Tu69a].) $S^{\neq}_{rat}$ *and* $S^{=}_{rat}$ *are properly included in* $S^{>}_{rat}$.

PROPOSITION 2. (See [Bu67].) *The class of regular languages is properly included in* $S^{=}_{rat} \bigcap S^{\neq}_{rat}$.

The rational stochastic classes are properly included in the class of context-sensitive languages (denoted CSL) (since $S^{>}_{rat} \subsetneq \text{Dspace}(n)$ [Di72] and $\text{Dspace}(n) \subset \text{Nspace}(n) = \text{CSL}$) but are not included in the class of context-free languages (denoted CFL); using the closure properties of $S^{=}_{rat}$, we can prove that the non-context-free language $\{a^n b^n c^n \mid n \in \mathbf{N}\}$ is included in $S^{=}_{rat}$ [Ma93].

On the other hand, there are context-free languages which are not contained in any of the rational stochastic classes: $\{x1y \mid x, y \in (0+1)^*,\ |x| = |y|\} \in \text{CFL} - S^{>}_{rat}$. (In fact this language is not even in $S^{>}$ [DS90, Ra92].)

It follows that rational stochastic classes do not fit neatly into the Chomsky's hierarchy. The next step is to compare them with space-bounded deterministic complexity classes. This problem is addressed in section 3.

Finally, we recall a useful result relating complexity classes defined by one-head one-way and one-head two-way probabilistic finite automata.

THEOREM 2. (See [Ka89].) $S^{>}_{rat} = 2\text{PFA}_{rat}$.

In other words, any 2pfa with rational transition probabilities can be simulated by a corresponding 1pfa with rational transition probabilities.

3. Space-efficient deterministic recognition of stochastic languages. In this section we compare the stochastic complexity classes we have defined so far ($S^{=}_{rat}$, $S^{\neq}_{rat}$, and $S^{>}_{rat}$) with classes of languages defined by space-bounded deterministic Turing machines. The main result is the (proper and optimal) inclusion of the class $S^{>}_{rat}$ in $\text{Dspace}(\log n)$. Lemma 3, used to compare integers given in terms of their residue representations, is of independent interest.

First, we present the background which is common to all the proofs in this section. For a 1pfa processing an input string w of length n, we have to compute its accepting probability and compare it with $1/2$. After some transformations, this task is equivalent to comparing two integers. Since the two integers are too large to fit in our working space, we work modulo relatively small primes, and we compare the residue representations of the two integers. In what follows, the representation of an integer modulo a set of primes (that is determined by the context) is called a *residue representation*. We denote by $X = (x_1, x_2, \ldots, x_n)$ (where $x_i = X \bmod p_i$, for all $i \in \{1, \ldots, n\}$) the residue representation of an integer X modulo the set of primes $\{p_1, p_2, \ldots, p_n\}$.

Now we present these steps in more detail. Let A be a 1pfa. Its transition probabilities are rational numbers. It follows that $M(w)$ has only rational elements for all $w \in \Sigma^*$. In constant space we can find the least common multiple (let us call it b) of the denominators of the transition probabilities and of the components of the initial-distribution vector. Without loss of generality, we suppose $\pi = (1, 0, \ldots, 0)$. We compute the accepting probability for the input string $w = w_1, \ldots, w_n$ of length n

$$p(w) = \pi M(w_1), \ldots, M(w_n)\eta_F^T$$

$$= (1/b^n)\pi M'(w_1), \ldots, M'(w_n)\eta_F^T,$$

where each $M'(w_i) = bM(w_i)$ has only integer elements from the interval [0,b]. Every element of $M'(w_i)$ can be computed in constant space. We have $p(w) = (1/b^n)p'(w)$ where $p'(w)$ is the word function generated by a 1Gpfa having $M'(w_i)$ as transition matrices $(p'(w) = \pi M'(w_1), \ldots, M'(w_n)\eta^T)$.

Comparing $p(w)$ with $1/2$ is equivalent to comparing $2p'(w)$ with b^n.

The values of $p'(w)$ and b^n are integers from the interval $[0, b^n]$, so $O(n)$ space is needed for storing them (too much); the standard solution is to compute the two numbers modulo relatively small primes and then to compare them using the Chinese Remainder Theorem. In this paper we work with the primes greater than 3. The reason why we avoid the prime 2 will be obvious from the proof of Lemma 1. (In fact this restriction is not essential because we can avoid using Lemma 1, but we have to modify Lemmas 2 and 3 accordingly [DMS94].)

Let us chose a constant c such that

$$\prod_{i=1}^{cn} p_i > 2^{cn} > 4b^n$$

and p_i is the ith prime starting with $p_1 = 3$. Then both $2p'(w)$ and b^n are natural numbers smaller than $\prod_{i=1}^{cn} p_i - 1$. We compute these numbers modulo $p_1, \ldots, p_{cn}$.

The advantage of using residue representations is that, at any given time, we do not have to store on the worktape all the residues, but only a finite number of them. Of course, when we need it, we have to be able to compute each such residue in a small amount of space. (In this way we use only $O(\log p_{cn}) = O(\log n)$ deterministic space.) Note that the ith prime can be found in $O(\log i)$ deterministic space. Also for each $k \in \{1, \ldots, cn\}$, the computation of $2p'(w) \bmod p_k$ requires $O(\log p_k) \in O(\log n)$ space since it is enough to keep on the work tape all the elements of the partial product matrix

$$M(w_1, \ldots, w_j) = \prod_{i=1}^{j} M(w_i) \bmod p_k, \text{ for } j \in \{1, \ldots, n\}.$$

For each $k \in \{1, \ldots, cn\}$ the computation of $b^n \bmod p_k$ uses only $O(\log n)$ space. It follows that the residue representations of $2p'(w)$ and b^n can be computed in $O(\log n)$ space, for one prime at a time.

Now we turn to the presentation of the main results of this section.

THEOREM 3. $\mathrm{S}^{=}_{\mathrm{rat}} \subsetneq \mathrm{Dspace}(\log \mathrm{n})$; $\mathrm{S}^{\neq}_{\mathrm{rat}} \subsetneq \mathrm{Dspace}(\log \mathrm{n})$ *and both inclusions are optimal.*

Proof. We prove the first relation. Since $\mathrm{Dspace}(\log n)$ is closed under complementation [Si80] and the complement of each language from $\mathrm{S}^{\neq}_{\mathrm{rat}}$ is in $\mathrm{S}^{=}_{\mathrm{rat}}$, it follows the second relation.

Let L be a language from $\mathrm{S}^{=}_{\mathrm{rat}}$, and let A be a 1pfa that recognizes L. Let w be an input word of length n and let $p(w)$ be the acceptance probability of w by A. It follows that

$$w \in L \Longleftrightarrow 1/2 = p(w) \Longleftrightarrow 2p'(w) = b^n.$$

We check the last equality (whose members are integers) modulo the first cn primes. If $2p'(w) \bmod p_i = b^n \bmod p_i$ for all $i \in \{1, \ldots, cn\}$ then $w \in L$; else $w \notin L$. It follows that the acceptance or rejection of w can be decided deterministically in $O(\log n)$ space, i.e., $L \in \mathrm{Dspace}(\log n)$. The time required by this simulation is $O(n^2)$. (We have assumed that each arithmetic operation between two $O(\log n)$-bit integers is done in constant time.)

Both inclusions are proper because the classes $\mathrm{S}^{=}_{\mathrm{rat}}$ and $\mathrm{S}^{\neq}_{\mathrm{rat}}$ are not closed under complementation. From the relations $\{a^n b^n \mid n \in \mathbf{N}\} \in \mathrm{Dspace}(\Omega(\log n))$ [LSH65] and $\{a^n b^n \mid n \in \mathbf{N}\} \in \mathrm{S}^{=}_{\mathrm{rat}}$ [Ma93], we obtain that both inclusions are optimal (i.e., it is not possible to include $\mathrm{S}^{=}_{\mathrm{rat}}$ and $\mathrm{S}^{\neq}_{\mathrm{rat}}$ in $\mathrm{Dspace}(o(\log n))$). □

In the rest of this section we show the stronger inclusion $\mathrm{S}^{>}_{\mathrm{rat}} \subsetneq \mathrm{Dspace}(\log \mathrm{n})$. To prove it we design a simulation of one-head probabilistic finite automata with rational transition probabilities by logspace deterministic Turing machines. The difference from the proof of Theorem 3 is that, at a given moment, we have to compare the numbers $2p'(w)$ and b^n using their residue representations. The standard procedure is to recompute the numbers from their residue representations (using the Chinese Remainder Theorem) and to compare them bit by bit. This approach was used by Jung [Ju84] who made crucial use of two results from circuit theory obtained by Reif [Re86] and Borodin [Bo77]. However, with this technique we can prove only $\mathrm{S}^{>}_{\mathrm{rat}} \subsetneq \mathrm{Dspace}(\log n \log\log n)$.

One observation about the method mentioned above is that we do not need to recompute the numbers (whose residue representations we already have) in order to decide which is larger. The comparison of these numbers can be done working directly on their residue representations. In what follows we focus on this idea and we prove $\mathrm{S}^{>}_{\mathrm{rat}} \subsetneq \mathrm{Dspace}(\log \mathrm{n})$ (Theorem 4). First we describe a technique to compare numbers, whose residue representations can be deterministically obtained in $O(\log n)$ space, using only $O(\log n)$ space. This result is contained in Lemmas 1–3.

LEMMA 1. *If N is an odd integer and $X, Y \in [0, N-1]$ are also integers, then $X \geq Y$ iff $(X-Y)$ has the same parity as $(X-Y) \bmod N$.*

Proof. It follows from the fact that N is odd and

$$(X-Y) \bmod N = \begin{cases} (X-Y) & if \quad X \geq Y \\ N+(X-Y) & if \quad X < Y. \end{cases}$$ □

LEMMA 2. *For every finite ascending sequence of primes $p_1, \ldots, p_n$, for every integer $X \in [0, \prod_{i=1}^{n} p_i)$, if the residue representation of X modulo these primes can be computed in $O(\log p_n)$ deterministic space, then $X \bmod N$ can be computed in $O(\log p_n + \log N)$ deterministic space, for every positive integer N.*

Proof. Let $X = (x_1, \ldots, x_n)$ be the residue representation of $X \in [0, \prod_{i=1}^{n} p_i)$ modulo the primes $p_1, \ldots, p_n$ (i.e., $x_i = X \bmod p_i$, for $i \in \{1, \ldots, n\}$), and let $M_n = \prod_{i=1}^{n} p_i$. Using the Chinese Remainder Theorem we obtain

$$X = \sum_{i=1}^{n} (x_i c_i)_{p_i} \frac{M_n}{p_i} - r \cdot M_n,$$

where r is a natural number, $c_i \in [0, P_i)$ is defined by $c_i \frac{M_n}{p_i} = 1 \pmod{p_i}$, and $(x_i c_i)_{p_i}$ denotes $x_i c_i \bmod p_i$, for $i = 1, n$. It follows that

$$r = \sum_{i=1}^{n} \frac{(x_i c_i)_{p_i}}{p_i} - \frac{X}{M_n},$$

where $0 \leq \frac{X}{M_n} < 1$ and $r = \lfloor \sum_{i=1}^{n} \frac{(x_i c_i)_{p_i}}{p_i} \rfloor \in [0, n)$.

The next step is to find deterministically r using small space. After computing r, using the Chinese Remainder Theorem we can easily compute $X \bmod N$ using only multiplications and additions modulo N.

In the rest of the proof we show how to space-efficiently find r. We compute each $t_i = \frac{(x_i c_i)_{p_i}}{p_i}$ with $(k+1)\lceil \log n \rceil$ exact digits and then we sum them up. We obtain the

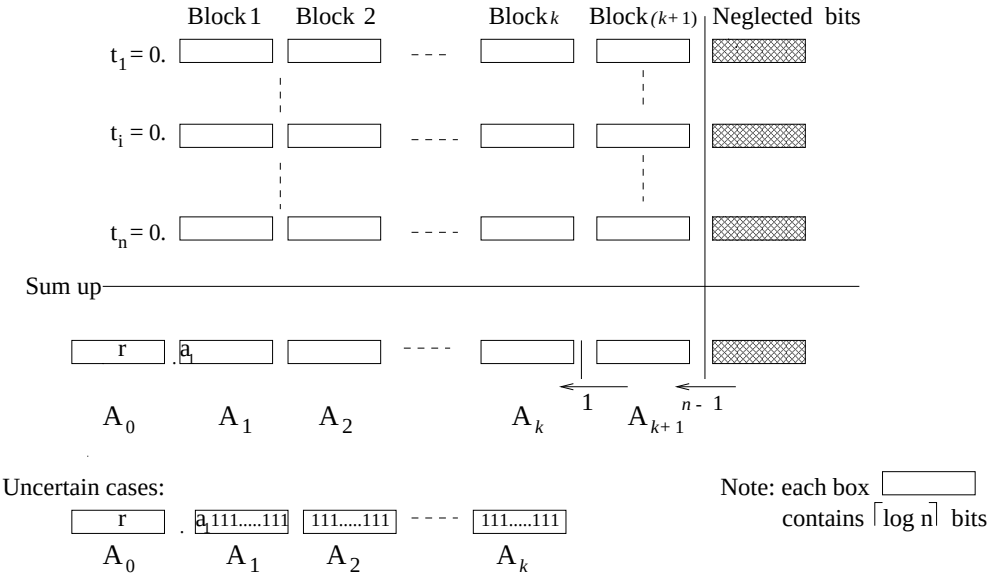


FIG. 1. *The "approximate" computation of r.*

number $A_0.A_1, \ldots, A_kA_{k+1}$, where A_i, $i \in \{0, \ldots, k+1\}$ are blocks of $\lceil \log n \rceil$ digits, and $A_0 \in [0, n)$ is the integer part of the sum. See Fig. 1. The value of k is obtained from the condition

$$\frac{M_n}{n^k} < \frac{M_{n-1}}{4} \Longleftrightarrow n^k > 4 \cdot p_n, \text{ for all } n \in \mathbf{N} \text{ greater than a threshold.}$$

(For example, if the sequence $p_1, p_2, \ldots, p_n$ contains the first n odd primes, then we can take $k = 2$.) In an exact computation of $\sum_{i=1}^{n} t_i$, it is possible to have a $(n-1)$-unit carry from the right of block A_{k+1}. Furthermore, block A_k may get at most one additional unit carry from block A_{k+1}. As a result, the bits in the blocks A_i, $i \in \{0, \ldots, k\}$, are exact or are affected by a one-unit carry to block A_k. If for all $i \in \{1, \ldots, k\}$ the blocks A_i have all their bits equal to 1, then a carry from block A_{k+1} does increment A_0, and this is the only situation when A_0 is affected by our computation error. We call such a situation an *uncertain* case. Therefore, we test whether all blocks A_i, $i \in \{1, \ldots, k\}$ have all their bits equal to 1; if not, we stop returning $r = A_0$, or else we continue with our investigation to decide whether $r = A_0$ or $r = A_0 + 1$. In this case we have

$$X \in \left[M_n - \frac{M_n}{n^k}, M_n\right] \text{ or } X \in \left[0, \frac{M_n}{n^k}\right]$$

(because our computation error is smaller than $1/n^k$). In order to treat both variants in a uniform way, we consider *negative* the numbers from the interval $(M_n/2, M_n)$ and *positive* the numbers from the interval $[0, M(n)/2)$. For every prime $p > 2$, let *smod* p be the function defined on integers and with values in the interval $(-p/2, p/2]$ such that

$$(X \; smod \, a) \bmod a = X \bmod a$$

for every integer X. Note that the function *smod* p is similar to the standard function *mod* p but it has values in the interval $(-p/2, p/2]$. It follows that the inclusion of X

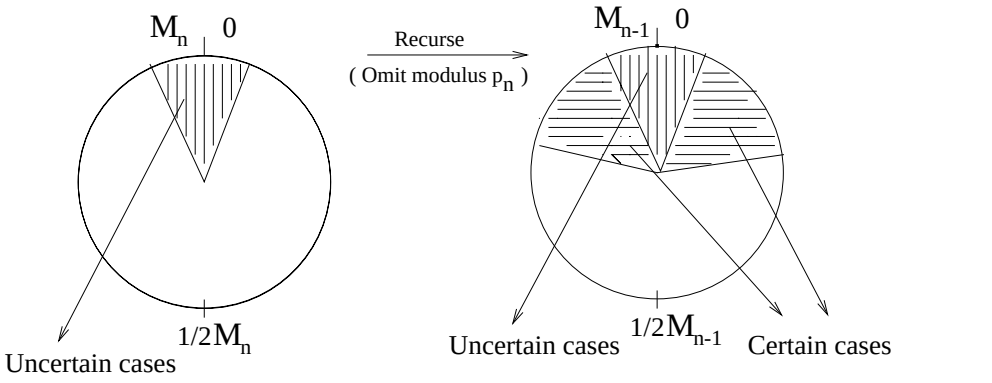


FIG. 2. *The reduction of uncertain cases.*

mentioned above is equivalent to

$$X \; smod \, M_n \in \left(-\frac{M_n}{n^k}, \frac{M_n}{n^k} \right).$$

We obtain that, in an uncertain case, our problem is reduced to finding the sign of $X \; smod \, M_n$.

This problem will be solved recursively. If $Y \; smod \, M_{n-1}$ is the number with the residue representation $(x_1, \ldots, x_{n-1})$, then the next recurrence holds:

$$X \; smod \, M_n = Y \; smod \, M_{n-1} + \alpha M_{n-1},$$

where α is an integer in the interval $(-p_n/2, p_n/2]$. From the fact that $X \; smod \, M_n \in (-\frac{M_n}{n^k}, \frac{M_n}{n^k}) \subset (-\frac{M_{n-1}}{4}, \frac{M_{n-1}}{4})$ (the value k has been computed to make this last inclusion true), and $Y \; smod \, M_{n-1} \in (-\frac{M_{n-1}}{2}, \frac{M_{n-1}}{2})$, it follows that $\alpha = 0$ in the previous recurrence. (This observation belongs to Dietz [Di93].) As a result,

$$X \; smod \, M_n = Y \; smod \, M_{n-1} \in \left(-\frac{M_{n-1}}{4}, \frac{M_{n-1}}{4} \right),$$

and we reduce the problem of computing the sign of $X \; smod \, M_n$ to computing the sign of $Y \; smod \, M_{n-1}$. If $Y \; smod \, M_{n-1} \in [0, \frac{M_{n-1}}{4})$, then $r = A_0 + 1$; else $r = A_0$. Note that this recursive step converts a large fraction of uncertain cases into certain cases. See Fig. 2.

If we still have an uncertain case, we continue the recursion as above. □

LEMMA 3. *Consider $p_1, p_2, \ldots, p_n$ strictly increasing odd primes and $M_n = \prod_{i=1}^{n} p_i$. If for two integers $X, Y \in [0, M_n - 1]$ their residue representations can be computed deterministically in $O(\log p_n)$ space, then the relations $X > Y, X = Y$, and $X < Y$ can be decided in $O(\log p_n)$ deterministic space.*

Proof. Let $X = (x_1, \ldots, x_n)$, $Y = (y_1, \ldots, y_n)$ be the residue representation of X and Y and let

$$Z = ((x_1 - y_1) \bmod p_1, \ldots, (x_n - y_n) \bmod p_n)$$

$$= (X - Y) \bmod M_n.$$

We compute the parity of X, Y, and Z using Lemma 2 with $N = 2$, we check whether $Z = 0$, and we apply Lemma 1 to compare X and Y. Every operation is done in $O(\log p_n)$ deterministic space. □

Observation 2. This lemma has a crucial role in proving the most important results of this paper. It gives us a space-efficient tool to compare large numbers given in terms of their residue representations. Another presentation and other applications of this technique can be found in [DMS94]. Davida and Litow, independently, propose another method to efficiently compare numbers given their residue representations [DL91]. The two methods turn out to have similar strength. Although one was expressed in the setting of deterministic space and the other in the setting of parallel time, they can easily be adapted to the other context. A more detailed comparison between them can be found in [DMS94].

THEOREM 4. $S^{>}_{\text{rat}} \subsetneqq \text{Dspace}(\log n)$ *and the inclusion is optimal.*

Proof. We work with odd primes and we follow the method presented at the beginning of this section until we have to compare two integers given (on demand) their residue representations. Then, we apply Lemma 3. The time required by this simulation is $O(n^3)$, under the assumption that arithmetic operations between $O(\log n)$-bit integers are done in constant time.

The inclusion is proper since $\{x1y \mid x, y \in (0+1)^*,\ |x| = |y|\} \in \text{Dspace}(\log n) - S^{>}_{\text{rat}}$. Also, it is optimal since $S^{=}_{\text{rat}}$ is optimally included in $\text{Dspace}(\log n)$ and $S^{=}_{\text{rat}} \subsetneqq S^{>}_{\text{rat}}$. □

This theorem improves the result of Dieu [Di72], $S^{>}_{\text{rat}} \subsetneqq \text{Dspace}(n)$.

COROLLARY 1. $2\text{PFA}_{rat} \subsetneqq \text{Dspace}(\log n)$ *and the inclusion is optimal.*

Proof. Use the equality $S^{>}_{\text{rat}} = 2\text{PFA}_{rat}$. □

This corollary improves the results of Wang [Wa92] ($2\text{BPFA}_{rat} \subsetneqq \text{Dspace}(n)$) and Jung [Ju84] ($\text{PrSpace}(\text{O}(1)) \subsetneqq \text{Dspace}(\log n \log\log n)$).

4. Space-efficient deterministic simulations of $o(\log\ n)$-space-bounded probabilistic Turing machines. The main result of this section is a space-efficient deterministic simulation of small-space-bounded probabilistic Turing machines. The simulation is presented for the more general setting of Markov chains. Our technique to compare a threshold with entries of the inverse of a given banded matrix is of independent interest.

We recall first some basic knowledge about space-bounded probabilistic computations. Let us consider the computation of a space-bounded PTM A on an input string x of length n. A configuration of A contains the state of its finite control, the description of the worktape content, and the position of heads on the worktape and on the input tape. Without loss of generality, we suppose that A has only one accepting configuration, one rejecting configuration, and that the computation halts when it enters these configurations. For an input x, the accepting probability of x is the probability that A enters the accepting configuration. The machine A accepts x if its accepting probability of x is larger than $1/2$.

In what follows, with the computation of A on an input string x, we associate a Markov process whose states are the possible configurations of A, and whose transition probabilities are given by the probabilities of A's moves. The Markov process has one starting state, one rejecting state, and one accepting state, according to the configurations of A. The accepting probability of x equals the long-run transition probability of the Markov process to move from the starting state into the accepting state. The Markov process is characterized by a state transition matrix. Each entry of this matrix represents the probability to move from one state to another state

in one step and corresponds to the transition probability of A to move from one configuration to another in one step. We call this matrix the *configuration transition matrix*. In order to compute easily the long-run transition from the initial state to the accepting state, we use a "simplified" configuration transition matrix in which we omit the rejecting state and some of the states that are not connected (directly or indirectly) to the accepting state. Obviously, the configuration transition matrix and the simplified configuration transition matrix have the same long-run transition probability from the initial state to the accepting state.

The acceptance decision of A on x is equivalent to comparing $1/2$ with a long-run transition probability in the associated Markov process. Using only logarithmic space, Simon [Si77, Si81] presents a way to reduce nondeterministically this problem to the problem of comparing $1/2$ with an element of a matrix inverse. This decision is obtained by a parallel-time-efficient (and also space-efficient, as follows from a result of Borodin [Bo77]) algorithm for matrix inversion due to Csanky [Cs76]. Jung [Ju84] finds a logspace deterministic reduction to reduce the acceptance decision of A on x to the problem of comparing $1/2$ with an entry of a matrix inverse, and he designs a space-efficient (and also parallel-time efficient) algorithm to solve the last problem in the particular case of matrices with small bandwidth.

In what follows we extend these previous techniques, and we present a more space-efficient deterministic simulation of small-space PTM's that makes crucial use of Lemmas 1–3.

THEOREM 5. *The languages recognized by $f(n) \in O(\log n)$ space-bounded probabilistic Turing machines can be recognized by deterministic Turing machines in space*

$$O(\min(c^{f(n)} \log n, \log n(f(n) + \log\log n))),$$

where c is a constant depending on the probabilistic Turing machine.[2]

Proof. Let A be an $f(n) \in O(\log n)$ space-bounded PTM processing an input string x of length n. By $p_A(x)$ we denote the acceptance probability of x by A. The number of possible configurations of A is

$$m = q(n+2)f(n)a_0^{f(n)} \in O(n^a),$$

where q is the number of states, a_0 is the size of the worktape alphabet of A, and a is a constant. (We remind the reader that A has one head on the input tape and one head on the worktape. Its input string is delimited by left and right end-markers.) We enumerate these configurations in increasing order of the input head's position. The computation of A on x is seen as a Markov process.

Let N be the m-by-m (simplified) configuration transition matrix associated with this Markov process in which the first row corresponds to the starting configuration of A (i.e., to c_1), the mth row, which corresponds to the accepting configuration (i.e., to c_m) is filled with 0, and the row (and the column) corresponding to the rejecting state is omitted. The bandwidth of N is $B \in O(f(n)a_0^{f(n)}) \subset O(c^{f(n)})$ (we can take $c = a_0 + \epsilon$ for arbitrarily small $\epsilon > 0$). Without loss of generality, we suppose that the elements of N are rational numbers all having the same denominator b. In the case of standard definition of PTM's b is 2, but we present the case with b natural, since the applicability of this result can be extended to other probabilistic devices whose computation can be represented by a Markov chain.

[2]More precisely, c depends on the size of the worktape alphabet of the probabilistic Turing machine.

The entry $(1, m)$ of the matrix sum $I + N + \cdots + N^k$ goes to $p_A(x)$ when k goes to infinity. Our problem is to space-efficiently approximate this entry. Also note that $\|N\| \leq 1$ and that the above matrix sum is not necessarily convergent. In what follows, we slightly modify N, using transformations computable deterministically in logarithmic space, to obtain the sequence of matrices Q, M, and finally $M - \Delta M$, so that $\|Q\| < 1$, $\|M\| < 1$, $\|M - \Delta M\| < 1$. Moreover, the sum $I + (M - \Delta M) + \cdots + (M - \Delta M)^k$ is convergent to F^{-1}, where $F = I - (M - \Delta M)$ is also easy to invert. The entry in the last column of the first row of F^{-1} will closely approximate the long-run transition probability from configuration c_1 into configuration c_m of the initial Markov process (i.e., $p_A(x)$). As a result, we can determine whether A accepts x by comparing that entry with $1/2$. This is equivalent to comparing two determinants whose elements are integers. We compute each of these determinants modulo relatively small primes and we use Lemma 3, where X and Y are the values of the two determinants.

In what follows we present the sequence of modifications of configuration transition matrices in steps. Recall that $\|X\|$ denotes the norm of the m-by-m matrix X, defined by $\|X\| = \max_i \sum_{j=1,m} |x_{ij}|$.

Step 1 (*obtaining* Q). Gill [Gi77] and Simon [Si81] proved that there is an integer d_1 independent of n, such that for every input x of length n the acceptance probability of x does not fall in the interval $[1/2, 1/2 + 1/b^{n^{d_1}}]$. (Such an integer can be found deterministically in $O(1)$ space. Using the technique from [RST82] (they used it to build "a probabilistic clock"), we find an integer $d_2 > d_1$ such that for every input x of length n, from each configuration c_i, A rejects with probability $1/b^{n^{d_2}}$ and enters in the next configuration c_j with probability $(1 - 1/b^{n^{d_2}}) \cdot n_{ij}$. Then the acceptance probability for x will decrease by a quantity less than $1/b^{n^{d_1}}$ and the set of recognized words is not changed. This modification of the computation of A corresponds to a modification of the associated Markov process and to a modification of the (simplified) configuration transition matrix N. (In fact, our goal is to modify the original Markov process corresponding to the computation of A on the input x. Some of these modifications could not correspond to any simple modification of the machine A.)

We obtain a configuration transition matrix Q of size m-by-m with its elements defined by $q_{ij} = (1 - 1/b^{n^{d_2}}) \cdot n_{ij}$. From $\|N\| \leq 1$ it follows $\|Q\| \leq 1 - 1/b^{n^{d_2}}$. In the associated Markov process the modified acceptance probability of x does not fall in the interval $[1/2, 1/2 + 1/b^{n^{d_3}}]$ for some integer $d_3 > d_1$ independent of n but dependent on d_1 and d_2.

So far we have obtained that $I - Q$ is nonsingular. In addition we would like to have the following property: the modified acceptance probability of x does not fall in an open interval containing $1/2$. In order to obtain this additional property in the next step we modify Q and obtain M.

Step 2 (*obtaining* M). We modify the Markov process by introducing a new state c_0 such that from c_0 the new Markov process goes into c_1 with probability $1 - 1/b^{n^{d_4}}$, where $d_4 > d_3$, and we consider c_0 its starting state. The long-run transition probability to move from state c_0 to state c_m is equal to $(1 - 1/b^{n^{d_4}})$ multiplied by the long-run transition probability to move from state c_1 to state c_m in the previous Markov process. Its values fall outside the interval $[1/2 - 1/b^{n^{d_5}}, 1/2 + 1/b^{n^{d_5}}]$, for some integer $d_5 > d_4$ independent of n, and are on the same side of $1/2$ as the original acceptance probability of x by A. We denote by M the new $(m + 1)$-by-$(m + 1)$

configuration transition matrix associated with this new Markov process. We recall that the last row of M (corresponding to the accepting configuration) contain only zeros. The matrix M has a bandwidth equal to B or $B+1$ and $\|M\| < 1-1/b^{n^{d_5}} < 1$. The matrix $I-M$ is nonsingular since it is *diagonally dominant*, i.e., for each row the absolute value of the diagonal element is larger than the sum of the absolute values of the other elements.

As a result, $I + M + \cdots + M^k + \cdots = (I-M)^{-1}$. Let $E = (I-M)^{-1}$. The element $e_{1,m+1}$ approximates the accepting probability of x by A in the following sense: if x is accepted by A, then $e_{1,m+1} > 1/2 + 1/b^{n^{d_5}}$, and if x is rejected by A, then $e_{1,m+1} < 1/2 - 1/b^{n^{d_5}}$. The integer d_5 is independent of n.

In the next step we apply one more transformation on M and obtain $M - \Delta M$, such that $I-(M-\Delta M)$ can be inverted space-efficiently using a Gaussian elimination. We achieve this by forcing $M-\Delta M$ to have nonnull elements on the diagonal situated immediately above the band of M.

Step 3 (*obtaining* $M-\Delta M$). We slightly modify M (and the Markov process) in a way that does not change the acceptance decision for x. We call ΔM the matrix having all its elements zeros with the exception of the elements on the diagonal immediately above the band of M, which are all equal to $1/b^{n^{d_6}}$, for an integer $d_6 > d_5$. Clearly, $\|\Delta M\| = 1/b^{n^{d_6}}$ and $\|M-\Delta M\| \le 1-1/b^{n^{d_5}} + 1/b^{n^{d_6}} < 1-1/2\cdot 1/b^{n^{d_5}}$. We compute the difference between $[I-(M-\Delta M)]^{-1}$ and $(I-M)^{-1}$ using a lemma of Banach [Go83]:

$$\|[I-(M-\Delta M)]^{-1} - (I-M)^{-1}\| \le \|\Delta M\| \cdot \|[I-(M-\Delta M)]^{-1}\| \cdot \|(I-M)^{-1}\|.$$

From $\|(I-M)^{-1}\| \le \|I\| + \|M\| + \cdots + \|M^k\| + \cdots \le \frac{1}{1-\|M\|}$ when $\|M\| < 1$, it follows that

$$\|(I-M)^{-1}\| \le \frac{1}{1-(1-1/b^{n^{d_5}})} = b^{n^{d_5}},$$
$$\|[I-(M-\Delta M)]^{-1}\| \le \frac{1}{1-(1-1/b^{n^{d_5}}+1/b^{n^{d_6}})} \le 2\cdot b^{n^{d_5}}.$$

Using Banach's lemma we obtain

$$\|[I-(M-\Delta M)]^{-1} - (I-M)^{-1}\| \le \frac{2\cdot b^{2n^{d_5}}}{b^{n^{d_6}}} < \frac{1}{b^{n^{d_5}}}$$

if $d_6 > d_5$ and if n is large enough. Let us denote $F = I-(M-\Delta M)$.

Based on the fact that $e_{1,m+1}$ closely approximates the acceptance probability of x in the sense described at the end of Step 2, and that the norm of $(E-F^{-1})$ is small enough, it follows that the element of F^{-1} from the entry $(1, m+1)$ lies on the same side of $1/2$ as $p_A(x)$. Computing this element is equivalent to computing the element y_1 from the system

$$F \cdot \begin{bmatrix} y_1 \\ \vdots \\ y_m \\ y_{m+1} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}.$$

Step 4 (*clear denominators*). Now, we transform the matrix F into an integer matrix. On the left, we multiply both sides of the previous system by the diagonal

matrix

$$F_1 = \begin{bmatrix} b^{n^{d_6}} & & & & & & \\ & \ddots & & & & & \\ & & \ddots & & & & \\ & & & b^{n^{d_6}} & & & \\ & & & & b^{n^{d_6}} & & \\ & & & & & 1 & \end{bmatrix}.$$

This yields the equivalent system

$$G \cdot \begin{bmatrix} y_1 \\ \vdots \\ y_m \\ y_{m+1} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}.$$

G has only integer elements, since d_6 exceeds the earlier constants, and it is nonsingular since F and F_1 are nonsingular. Its bandwidth is at most one larger than the bandwidth of N, and its elements on its highest nonnull diagonal are all 1. We have to find y_1 and how it compares with $1/2$. The element y_1 is the entry $(1, m+1)$ of G^{-1}, so its value is $y_1 = \frac{u}{v}$, where $u = (-1)^{(m+1)+1}\Delta_{(m+1,1)}$, $v = \det G$, u and v are both integers, and $u/v > 1/2$ iff $2u > v$. $\Delta_{(m+1,1)}$ is the minor $(m+1, 1)$ of G, and is the determinant of a matrix having the same structure as G; so our problem is reduced to comparing the determinants of two matrices that have the same type as G.

Step 5 (*Gaussian elimination*). We show how to compute the determinant of G. The determinants giving u and v can be computed in a similar way. First, we do a partial Gaussian elimination of the elements under the highest nonnull diagonal (now filled with 1's). Next we have to compute the determinant of a $O(B)$-by-$O(B)$ integer matrix G_2 (see Fig. 3) that can be done in $O(\log B(\log B + \log\log P))$ deterministic space, where P is the largest number used in that computation.

We comment now on different ways to do the Gaussian elimination and to compute the elements of the matrices G_1 and G_2. The first remark is that we can compute the elements of G_1 and G_2 column by column. For a given column, each element can be computed by a linear recurrence involving a constant number of elements belonging to the same column and situated in the next upper $(B+1)$ positions above it. We have to compute the first $m+1$ elements generated by this recurrence. (Note that the operations involved in the recurrence are additions, subtractions, and multiplications.) We indicate two ways to solve this recurrence, whose space-efficiencies depend on the magnitude of B.

- One way is to always keep the last $(B+1)$ elements on the worktape, and, when computing a new element, to replace the oldest element. The space used in this approach is $O(B \cdot \log P)$.
- The other way is to solve the recurrence in parallel using a circuit model. We recall the result of Stone [St73], which states that obtaining the first m numbers satisfying a linear recurrence of order B is equivalent to multiplying m B-by-B matrices, and this is equivalent to performing $\log m$ iterations, where each iteration involves parallel multiplications of B-by-B matrices. Matrix multiplication can be done by logspace-uniform (polynomial-size) boolean circuits of depth $O(\log B + \log\log P)$, where P is the largest number involved in the computation. As a result, we can solve the linear recurrence using

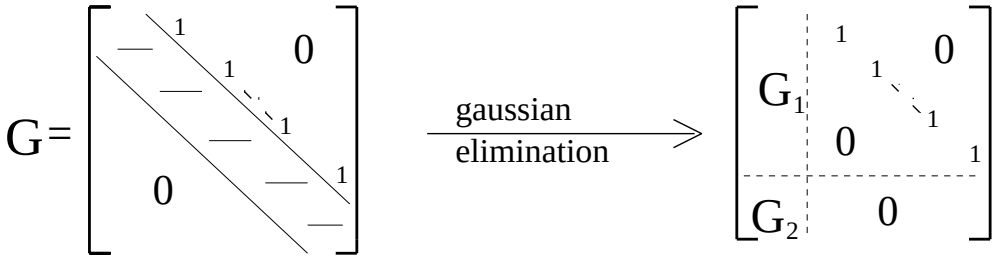


FIG. 3. *Gaussian elimination.*

logspace-uniform boolean circuits of depth $O(\log m(\log B + \log\log P))$, which can be simulated in $O(\log m(\log B + \log\log P))$ deterministic space using a result of Borodin [Bo77]. Essentially, Jung found another variant of this second method [Ju84].

In summary, the acceptance or rejection of x is equivalent to deciding whether y_1 is larger or smaller than $1/2$, which is equivalent to deciding which of the integers $2u$ and v is larger. The magnitude of these two numbers is $O(m(b^{n^{d_6}})^m) = O(b^{n^{d_6+a+1}})$ (recall that $m \in O(n^a)$). Let $h \in \Theta(n^{d_6+a+1})$. We have $\max(2u, v) < \prod_{i=1}^{h} p_i$, where p_i, $i \in \{1, \ldots, h\}$ are the first h primes starting with $p_1 = 3$. We can compute $2u$ and v modulo these h primes and then compare them using Lemma 3. Their computation modulo a prime p_i can be done deterministically, using the previous remarks, in $O(\log m + \min(B \log p_i, \log m(\log B + \log\log p_i)))$ space. Also note that $\log p_h \in O(\log n)$ (because $p_h \in O(h \log h)$), $\log m \in O(\log n)$, and $B = O(c^{f(n)})$. □

THEOREM 6. *The languages recognized by $f(n) \in o(\log\log\log n)$ space-bounded probabilistic Turing machines can be recognized by deterministic Turing machines in space $O(c^{f(n)}(\log n))$, where c is a constant depending on the probabilistic Turing machine.*

This theorem improves the following result of Jung [Ju84]:

$$\mathrm{PrSpace}(f(n)) \subset \mathrm{Dspace}(\log n(f(n) + \log\log n))$$

for $f(n) \in o(\log\log\log n)$.

5. Discussion and open problems. This article presents space-efficient derandomized (i.e., deterministic) simulations of probabilistic automata. Depending on the amount of space used by the simulated probabilistic automaton, we obtain results at three different levels:

- Probabilistic (one-head one-way and one-head two-way) finite automata (with rational transition probabilities) can be simulated deterministically in $O(\log n)$ space. This simulation is optimal and represents a significant improvement over the previously reported simulations [Di72, Ju84, Wa92]. It follows as a corollary that the languages recognized by these automata have simple characterizations in extended first-order logics. (See [Im87] for some connections among space-bounded complexity classes and first-order logic with ordering and closure operators.)
- Probabilistic Turing machines with space bound $f(n) \in o(\log\log\log n)$ are simulated deterministically in $O(c^{f(n)} \log n)$ space, where c is a constant depending on the size of the worktape alphabet of the simulated machine. This

improves a simulation of Jung, which uses $O(\log n \log\log n)$ space independently of the size of $f(n)$ [Ju84]. The existence of languages with probabilistic space complexities at this level was pointed out by Freivalds [Fr81].[3] In the nondeterministic case, on the other hand, there are no such languages since there is a space complexity gap below $\log\log n$ [SHL65].

- Probabilistic Turing machines with space bound $f(n)$ between $\Theta(\log\log\log n)$ and $\Theta(\log n)$ can be simulated deterministically in $O(\log n(f(n) + \log\log n))$ space. Our new simulation uses the same space as Jung's [Ju84]. When $f(n)$ is between $\Theta(\log\log n)$ and $\Theta(\log n)$ these simulations generalize Monien and Sudborough's deterministic simulation of space-bounded nondeterministic Turing machines [MS82].

Our simulations are presented in the more general setting of Markov chains, so they may have other applications as well. We also design a space-efficient method to compare a threshold with entries of the inverse of banded matrices. One tool for proving these results is a space-efficient (and also parallel-time-efficient) technique to deterministically compare numbers given in terms of their residue representations.

We turn now to some open problems.

The main limitation in the proof of Theorem 5 is that we are able to use only the bandwidth of the matrix, but not its sparse structure, within the band. We are looking for more sophisticated algorithms for inverting matrices, able to take advantage of bandwidth combined with sparseness. Interestingly, for the problem of iterated matrix product, to which matrix inversion is efficiently reducible [Co85], there is known an approximation technique that takes advantage of the matrix sparseness within each row [SZ95]. This technique uses pseudorandom generators for space-bounded bounded-error probabilistic computation [Ni90, Ni92]. However, it does not provide tight approximations and, consequently, it can be used for simulating only *bounded*-error probabilistic computation. It may be possible to combine the advantages of both techniques.

Another question is how to take further advantage of the fact that we can do small modifications on the configuration transition matrices when inverting them.

Acknowledgments. I am very grateful to Joel Seiferas and Paul Dietz for many valuable discussions on this subject. I also thank them for carefully reading an earlier version of this paper and suggesting significant improvements. I also thank Helmut Jürgensen for his comments on a preliminary version of this paper and Marius Zimand and Bruce Litow for bringing to my attention the paper by Davida and Litow [DL91].

REFERENCES

[Bo77] A. Borodin, *On relating time and space to size and depth*, SIAM J. Comput., 6 (1977), pp. 733–744.

[BCP83] A. Borodin, S. Cook, and N. Pippenger, *Parallel computation for well-endowed rings and space-bounded probabilistic machines*, Inform. and Control, 48 (1983), pp. 113–136.

[Bu67] R. Bukharaev, *On the representability of events in probabilistic automata*, Prob. Methods and Cybernetics, V, Kazan, 1967, pp. 7–20 (in Russian).

[Bu85] R. Bukharaev, *Fundamentals of the Theory of Probabilistic Automata*, Nauka, Moscow, 1985 (in Russian).

[CK94] K. Culik and J. Karhumaki, *Finite automata computing real functions*, SIAM J. Comput., 23 (1994), pp. 789–814.

[3] Actually, Freivalds used a different notion of space-bounded probabilistic complexity class, but our simulation can also be adapted to his case.

[Co85] S. A. Cook, *A taxonomy of problems with fast parallel algorithms*, Inform. and Comput., 64 (1985), pp. 2–22.

[Cs76] L. Csanky, *Fast parallel matrix inversions algorithms*, SIAM J. Comput., 5 (1976), pp. 618–623.

[Di71] P. D. Dieu, *On a class of stochastic languages*, Z. Math. Logik Grundlagen Math. Bd., 17 (1971), pp. 421–425.

[Di72] P. D. Dieu, *On a necessary condition for stochastic languages*, Electron. Inform. Kybernetik EIK, 8/10 (1972), pp. 575–588.

[Di93] P. Dietz, personal communication, 1993.

[DL91] G. I. Davida and B. Litow, *Fast parallel arithmetic via modular representation*, SIAM J. Comput., 20 (1991), pp. 756–765.

[DMS94] P. Dietz, I. I. Macarie, and J. I. Seiferas, *Bits and relative order from residues, space efficiently*, Inform. Process. Lett., 50 (1994), pp. 123–128.

[DS90] C. Dwork and L. Stockmeyer, *A time complexity gap for two-way probabilistic finite-state automata*, SIAM J. Comput., 19 (1990), pp. 1011–1023.

[Eb89] W. Eberly, *Very fast parallel matrix and polynomial arithmetic*, SIAM J. Comput., 18 (1989), pp. 955–976.

[Eb95] W. Eberly, *Fast parallel band matrix arithmetic*, Inform. and Comput., 116 (1995), pp. 117–127.

[Fr81] R. Freivalds, *Probabilistic two-way machines*, in Proc. 6th Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci., 118, Springer-Verlag, New York, 1981, pp. 33–45.

[Fr91] R. Freivalds, *Complexity of probabilistic versus deterministic automata*, Baltic Computer Science, Selected Papers, Lecture Notes in Comput. Sci. 502, Springer-Verlag, Berlin, 1991, pp. 565–613.

[Gi77] J. Gill, *Computational complexity of probabilistic Turing machines*, SIAM J. Comput., 6 (1977), pp. 675–695.

[Go83] G. Golub and C. Van Loan, *Matrix Computations,* The Johns Hopkins University Press, Baltimore, MD, 1983.

[HU79] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison–Wesley, Reading, MA, 1979.

[Im87] N. Immerman, *Languages that capture complexity classes*, SIAM J. Comput., 16 (1987), pp. 760–778.

[Ju81] H. Jung, *Relationships between probabilistic and deterministic tape complexity*, in Proc. 10th Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 118, Springer-Verlag, New York, 1981, pp. 339–346.

[Ju84] H. Jung, *On probabilistic tape complexity and fast circuits for matrix inversion problems*, 11th Internat. Coll. on Automata, Languages and Programming, (ICALP 1984), Lecture Notes in Comput. Sci. 172, Springer-Verlag, New York, pp. 281–291.

[Ka89] J. Kaneps, *Stochasticity of the languages recognizable by* 2*-way finite probabilistic automata*, Diskret. Mat., 1 (1989), pp. 63–77 (in Russian).

[Ku73] Yu. I. Kuklin, *Two-way probabilistic automata*, Avtomat. i Vychisl. Tekhn., 5 (1973), pp. 35–36 (in Russian).

[LR86] F. Leighton and R. Rivest, *Estimating a probability using finite memory*, IEEE Trans. Inform. Theory, 6 (1986), pp. 733–742.

[LSH65] P. M. Lewis II, R. Stearn, and J. Hartmanis, *Memory bounds for recognition of context-free and context-sensitive languages*, in IEEE Conference Record on Switching Circuit Theory and Logical Design, 1965, IEEE Computer Science Press, Los Alamitos, CA, pp. 191–202.

[Ma93] I. Macarie, *Closure Properties of Stochastic Languages,* Technical Report TR441, Dept. of Comp. Science, University of Rochester, Rochester, NY, 1993.

[Ma95] I. I. Macarie, *Decreasing the bandwidth of a configuration transition matrix*, Inform. Process. Lett., 53 (1995), pp. 315–320.

[MS82] B. Monien and I. H. Sudborough, *On eliminating nondeterminism from Turing machines which use less than logarithm worktape space*, Theoret. Comput. Sci., 21 (1982), pp. 237–253.

[Ni90] N. Nisan, *Pseudorandom generators for space-bounded computation*, Proc. 22nd Annual ACM Symposium on the Theory of Computing, ACM, New York, 1990, pp. 204–212.

[Ni92] N. Nisan, $RL \subseteq SC$, in Proc. 24th Annual ACM Symposium on the Theory of Computing, ACM, New York, 1992, pp. 619–623.

[NZ93] N. Nisan and D. Zuckerman, *More deterministic simulation in logspace*, in Proc. 25th Annual ACM Symposium on the Theory of Computing, ACM, New York, 1993, pp. 235–244.

[Pa71] A. Paz, *Introduction to probabilistic automata,* in Computer Science and Applied Mathematics, Academic Press, New York, 1971.

[Ra63] M. O. Rabin, *Probabilistic automata*, Inform. and Control, 6 (1963), pp. 230–244.

[Re86] J. Reif, *Logarithmic depth circuits for algebraic functions*, SIAM J. Comput., 15 (1986), pp. 231–242.

[Ra92] B. Ravikumar, *Some Observations on 2-way Probabilistic Finite Automata*, Technical Report TR92-208, Dept. of Comp. Science and Statistics, Univ. of Rhode Island, Kingston, RI, 1992.

[RST82] W. Ruzzo, J. Simon, and M. Tompa, *Space-bounded hierarchies and probabilistic computation*, in Proc. 14th Annual ACM Symposium on Theory of Computing, ACM, New York, 1982, pp. 215–223; also in J. Comput. System Sci., 28 (1984), pp. 216–230.

[Sa70] W. J. Savitch, *Relationships between nondeterministic and deterministic tape complexity*, J. Comput. System Sci., 4 (1970), pp. 177–192.

[SHL65] R. E. Stearns, J. Hartmanis, and P. M. Lewis II, *Hierarchies of memory limited computations*, in IEEE Conf. Record on Switching Circuit Theory and Logical Design, IEEE Computer Society Press, Los Alamitos, CA, 1965, pp. 179–190.

[Si77] J. Simon, *On the difference between one and many*, in Proc. 4th Internat. Coll. on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 52, Springer-Verlag, New York, 1977, pp. 480–491.

[Si81] J. Simon, *Space-bounded probabilistic Turing machine complexity classes are closed under complement*, in Proc. 13th Annual ACM Symposium on Theory of Computing, ACM, New York, 1981, pp. 158–167.

[Si80] M. Sipser, *Halting space-bounded computation*, Theoret. Comput. Sci. 10 (1980), pp. 335–338.

[SS78] A. Salomaa and M. Soittola, *Automata-theoretic Aspects of Formal Power Series*, Springer-Verlag, Berlin, 1978.

[St66] P. Starke, *Stochastische Ereignisse und Wortmengen*, Z. Math. Logik Grundlagen Math. Bd., 12 (1966), pp. 61–68.

[St73] H. Stone, *An efficient parallel algorithm for the solution of a tridiagonal linear system of equations*, J. ACM, 20 (1973), pp. 27–38.

[SZ95] M. Saks and S. Zhou, $RSPACE(S) \subseteq DSPACE(S^{3/2})$, in Proc. 36th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 344–353.

[Tu68] P. Turakainen, *On stochastic languages*, Inform. and Control, 12 (1968), pp. 304–313.

[Tu69a] P. Turakainen, *On languages representable in rational probabilistic automata*, Ann. Acad. Sci. Fenn. Ser. A I, 439 (1969).

[Tu69b] P. Turakainen, *Generalized automata and stochastic languages*, Proc. Amer. Math. Soc., 21 (1969), pp. 303–309.

[Tu71] P. Turakainen, *Some closure properties of the family of stochastic languages*, Inform. and Control, 18 (1971), pp. 253–256.

[Wa92] J. Wang, *A note on two-way probabilistic automata*, Inform. Process Lett., 43 (1992), pp. 321–326.

SIAM J. COMPUT.
Vol. 27, No. 2, pp. 466–490, April 1998
© 1998 Society for Industrial and Applied Mathematics
008

EFFICIENT MATRIX CHAIN ORDERING IN POLYLOG TIME*

PHILLIP G. BRADFORD†, GREGORY J. E. RAWLINS‡, AND GREGORY E. SHANNON§

Abstract. The matrix chain ordering problem is to find the cheapest way to multiply a chain of n matrices, where the matrices are pairwise compatible but of varying dimensions. Here we give several new parallel algorithms including $O(\lg^3 n)$-time and $n/\lg n$-processor algorithms for solving the matrix chain ordering problem and for solving an optimal triangulation problem of convex polygons on the common CRCW PRAM model. Next, by using efficient algorithms for computing row minima of totally monotone matrices, this complexity is improved to $O(\lg^2 n)$ time with n processors on the EREW PRAM and to $O(\lg^2 n \lg\lg n)$ time with $n/\lg\lg n$ processors on a common CRCW PRAM. A new algorithm for computing the row minima of totally monotone matrices improves our parallel MCOP algorithm to $O(n \lg^{1.5} n)$ work and polylog time on a CREW PRAM. Optimal log-time algorithms for computing row minima of totally monotone matrices will improve our algorithm and enable it to have the same work as the sequential algorithm of Hu and Shing [*SIAM J. Comput.*, 11 (1982), pp. 362–373; *SIAM J. Comput.*, 13 (1984), pp. 228–251].

Key words. parallel, dynamic programming, matrix chain ordering, optimization

AMS subject classifications. 90C39, 68Q25, 03D15

PII. S0097539794270698

1. Introduction.

The design of efficient parallel algorithms for problems with elementary serial dynamic programming solutions has been the focus of much recent research. These problems include string editing [2, 5, 30], context-free grammar recognition [35, 37], and optimal tree building [7, 32]. Polylog time parallel algorithms for solving these problems use new approaches, since straightforward parallelization of sequential dynamic programming algorithms produce very slow (linear-time) parallel algorithms. Many efficient parallel algorithms designed to date rely on *monotonicity conditions* to give divide-and-conquer schemes. By "efficient" we mean that the processor-time product is within a polylog factor of the best sequential time.

The *matrix chain ordering problem* (MCOP) is to find the cheapest way to multiply a chain of n matrices, where the matrices are pairwise compatible but of varying dimensions. This problem can be found in many classic textbooks on parallel and sequential algorithms, such as [3, 18]. The MCOP is often the focus of dynamic programming research and pedagogy because of its amenability to an elementary dynamic programming solution. There has been significant sequential and parallel work on the MCOP [11, 17, 19, 20, 21, 25, 26, 27, 28, 29, 38, 40, 41, 39, 42, 43, 44] and a related convex polygon triangulating problem. However, until recently none of this work has given an efficient (linear-processor) polylogarithmic-time algorithm for the

*Received by the editors July 6, 1994; accepted for publication (in revised form) February 13, 1996. An extended abstract of this paper appeared in the *Proceedings of the 8th Annual IEEE International Parallel Processing Symposium*, Cancun, Mexico, H. J. Siegel, ed., IEEE Computer Society Press, Los Alamitos, CA, pp. 234–241. This paper also represents part of the first author's dissertation.

http://www.siam.org/journals/sicomp/27-2/27069.html

†Department of Computer Science, Indiana University, 215 Lindley Hall, Bloomington, IN 47405 (bradford@cs.indiana.edu). Current address: Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany (pbradford@bfm.com).

‡Department of Computer Science, Indiana University, 215 Lindley Hall, Bloomington, IN 47405 (rawlins@cs.indiana.edu).

§Department of Computer Science, Indiana University, 215 Lindley Hall, Bloomington, IN 47405 (shannon@cs.indiana.edu). Current address: 2501 N. Loop Drive, ISU Research Park, Ames, IA 50010 (shannon@neb.infostructure.com).

MCOP. (See Bradford [11].) Recently in [40] and [41] Ramanan independently gave an $O(\lg^4 n)$-time and n-processor algorithm for solving the MCOP on the CREW PRAM. Moreover, in [15] we gave a $O(\lg^4 n)$-time and $n/\lg n$ processor algorithm for solving the MCOP on the common-CRCW PRAM.

Algorithm-design paradigms often aid the design of efficient sequential algorithms. However, some algorithm-design paradigms may not lead to efficient parallel algorithms. In particular, some variations of the greedy paradigm appear to be inherently sequential [4]. This highlights the significance of research in parallel dynamic programming.

1.1. Main results of this paper. Our approach follows [11], recasting the MCOP as a shortest path problem in a graph modeling a dynamic programming table. (In fact, this paper is an update of a part of [10] and a revision of [15]; see also the first author's dissertation [12].) This graph has $O(n^2)$ nodes and with an all-pairs shortest paths algorithm finding a shortest path in this graph results in a $n^6/\lg n$ processor MCOP algorithm. Reducing the number of nodes to $O(n)$ using a tree decomposition and applying an all-pairs shortest path algorithm gives an $n^3/\lg n$ processor and polylog-time algorithm.

In this paper, we convert the successive applications of the brute force all-pairs shortest paths algorithm to successive applications of parallel partial prefix and binary search algorithms. As in the $n^3/\lg n$-processor algorithm, the applications of the prefix and binary search algorithms are controlled by a rake-compress paradigm operating on a tree-based decomposition of the original graph. All of this results in a polylog-time ($O(\lg^3 n)$) and linear-processor ($n/\lg n$) parallel algorithm for the MCOP on the common-CRCW PRAM. This improves our result of $O(\lg^4 n)$ time and $n/\lg n$ processors of [15]. In addition, using efficient algorithms for computing row minima on totally monotone matrices, our algorithm can run in $O(\lg^2 n \lg\lg n)$ time using $n/\lg\lg n$ processors on a common-CRCW PRAM or in $O(\lg^2 n)$ time using n processors on an EREW PRAM. Using the most efficient polylog-time parallel algorithms for computing row minima on totally monotone matrices, our algorithm can run in $O(\lg^{1.5} n \lg\lg n)$ time using $O(n\sqrt{\lg n})$ work on a common-CRCW PRAM or in $O(\lg^{2.5} n\sqrt{\lg\lg n})$ time using $O(n\sqrt{\lg n \lg\lg n})$ processors on an EREW PRAM. See Bradford, Fleischer, and Smid [14] for the most efficient polylog-time parallel algorithms to date for computing row minima in totally monotone matrices.

1.2. Previous results. Elementary dynamic programming algorithms sequentially solve the matrix chain ordering problem in $O(n^3)$ time; see [3, 18]. Several recent textbooks on the design and analysis of parallel algorithms discuss the MCOP; see [36, 23]. However, the best sequential solution of the MCOP is Hu and Shing's $O(n \lg n)$ algorithm [28, 29]. (Much work has been done on lower bounds on the MCOP and related problems; see [38, 39, 13].) Using straight-line arithmetic programs, Valiant et al. [43] showed that many classical optimization problems with efficient sequential dynamic programming solutions are in $\mathcal{NC}$. Their algorithms require $\Theta(\lg^2 n)$ time and n^9 processors. Using pebbling games, Rytter [42] gave more efficient parallel algorithms for a similar class of optimization problems costing $O(\lg^2 n)$ time with $n^6/\lg n$ processors. In [11], an algorithm was given that takes $O(\lg^3 n)$ time and $n^3/\lg n$ processors, and [20] gave an algorithm that takes $O(\lg^3 n)$ time and $n^2/\lg^3 n$ processors. In [40], Ramanan gives an extended abstract of an n-processor and $O(\lg^4 n)$-time CREW PRAM algorithm for solving the MCOP which came after our buggy version in [10]; his full version appears in [41]. A full version of our $n/\lg n$-processor and $O(\lg^4 n)$-time algorithm described and improved upon in this paper

appears in [15]. In addition, there are serial and parallel approximation algorithms for the MCOP [11, 17, 19, 27].

1.3. Structure of the paper. In section 2 we briefly review the interpretation of the MCOP as a shortest path graph problem from [11] and then summarize the $n^3/\lg n$-processor algorithm. In section 3 we isolate this algorithm's $n^3/\lg n$-processor bottlenecks. The $n^3/\lg n$-processor cost of these bottlenecks is from an all-pairs shortest paths algorithm. In section 4 we show how to replace the all-pairs shortest path algorithm with parallel prefix and an all-pairs *comparison* algorithm. In section 5 we replace the all-pairs comparison algorithm with applications of parallel prefix and binary search. Finally, it is shown that the key problems solved in section 4, and more efficiently in section 5, can be solved by finding the row minima of a totally monotone matrix.

2. An $O(\lg^3 n)$ time and $n^3/\lg n$ processor MCOP algorithm. This section contains a brief review of the polylog-time and $n^3/\lg n$-processor MCOP algorithm from [11].

Let T be an $n \times n$ dynamic programming table for the matrix chain ordering problem. It has entries $T[i,k]$ representing the cheapest cost of the matrix product $M_i \bullet \cdots \bullet M_k$. For any such T there is a graph D_n where the cost of a shortest path to node (i,k), denoted $sp(i,k)$, is the same as the final value of $T[i,k]$. Given a chain of n matrices, finding a shortest path from $(0,0)$ to $(1,n)$ in D_n solves the MCOP [11].

The weighted digraph D_n has vertices in the set, $\{(i,j) : 1 \le i \le j \le n\} \cup \{(0,0)\}$ and edges

$$\{(i,j) \to (i,j+1) : 1 \le i \le j < n\} \cup \{(i,j) \uparrow (i-1,j) : 1 < i \le j \le n\}$$
$$\cup \{(0,0) \nearrow (i,i) : 1 \le i \le n\},$$

known as *unit* edges, together with the edges

$$\{(i,j) \Longrightarrow (i,t) : 1 < i < j < t \le n\} \cup \{(s,t) \Uparrow (i,t) : 1 \le i < s < t \le n\},$$

known as *jumpers*; see the jumper from $(1,2)$ to $(1,4)$ in Figure 1. The unit edge $(i,j) \to (i,j+1)$ represents the product $(M_i \bullet \cdots \bullet M_j) \bullet M_{j+1}$ and weighs $f(i,j,j+1) = w_i w_{j+1} w_{j+2}$, which is taken as the cost of multiplying a $w_i \times w_{j+1}$ matrix and a $w_{j+1} \times w_{j+2}$ matrix. Similarly, the unit edge $(i,j) \uparrow (i-1,j)$ represents the product $M_{i-1} \bullet (M_i \bullet \cdots \bullet M_j)$ and costs $f(i-1,i-1,j) = w_{i-1} w_i w_{j+1}$. A shortest path to (i,k) through the jumpers $(i,j) \Longrightarrow (i,k)$ and $(j+1,k) \Uparrow (i,k)$ represents the product $(M_i \bullet \cdots \bullet M_j) \bullet (M_{j+1} \bullet \cdots \bullet M_k)$, and these jumpers weigh $sp(j+1,k) + f(i,j,k)$ and $sp(i,j) + f(i,j,k)$, respectively, where $sp(j+1,k)$ is the cost of a shortest path to node $(j+1,k)$ and $f(i,j,k) = w_i w_{j+1} w_{k+1}$. The jumper $(i,j) \Longrightarrow (i,t)$ is of length $t-j$. See Figure 1.

Using this model the MCOP can be solved in polylog time with $n^6/\lg n$ processors by using an all-pairs shortest path algorithm and exploiting the following theorem.

THEOREM 1 (Duality Theorem [11]). *If a shortest path from* $(0,0)$ *to* (i,k) *contains the jumper* $(i,j) \Longrightarrow (i,k)$*, then there is a* dual *shortest path containing the jumper* $(j+1,k) \Uparrow (i,k)$.

Furthermore, using a tree decomposition of D_n and an all-pairs shortest path algorithm, the MCOP was solved in polylog time using $n^3/\lg n$ processors [11].

2.1. Matrix dimensions as nesting levels of matching parentheses. The next four subsections show that using the list of matrix dimensions as nesting levels of

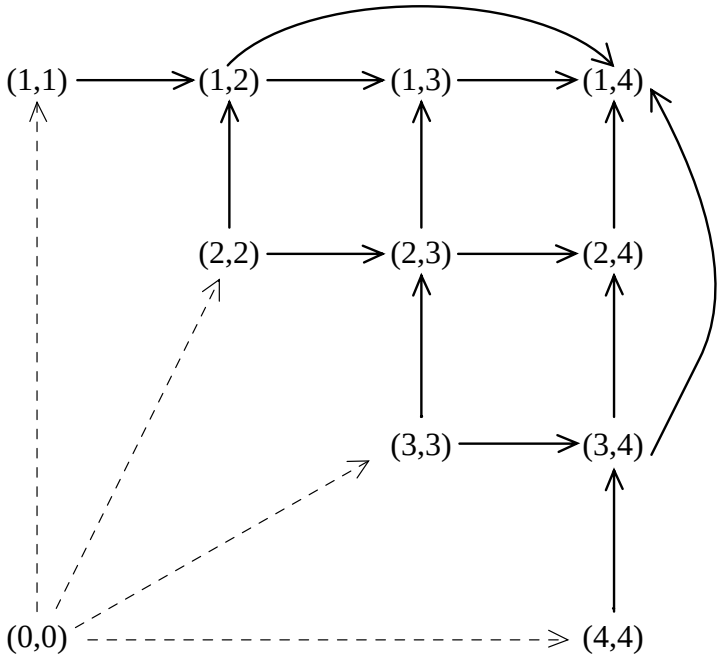


FIG. 1. *The weighted graph D_4.*

matching parentheses gives a tree decomposition of D_n that leads to efficient solutions of the MCOP.

Given an associative product with the level of each parenthesis known, for each parenthesis find its matching parenthesis by solving the all nearest smaller value (ANSV) problem [8, 9]: given weights $w_1, w_2, \ldots, w_n$, for each w find the indices, if they exist, of the nearest proceeding and succeeding weights both less than w. Let's call this pair of indices, if they exist, an ANSV *match*. That is, for each w the problem is to find the largest j where $1 \leq j < i$, and find the smallest k where $i < k \leq n$, so that $w_j < w_i$ and $w_k < w_i$, if such values exist. In D_n, (i, k) is a *critical node* if $[w_i, w_{k+1}]$ is an ANSV match.

By solving the ANSV problem we can compute all critical nodes of D_n. The bottom of Figure 2 depicts a list of matrix dimensions (called weights) and dashed lines representing four key ANSV matches. The four corresponding critical nodes are circled in D_n.

In our nomenclature, [8] shows that the following theorem holds.

THEOREM 2. *Computing all critical nodes costs $O(\lg n)$ time with $n/\lg n$ processors or in $O(\lg \lg n)$ time using $n/\lg \lg n$ processors on the common-CRCW PRAM.*

In addition, [16, 34] give the following theorem.

THEOREM 3. *Computing all critical nodes costs $O(\lg n)$ time with $n/\lg n$ processors on the EREW PRAM.*

Two critical nodes on the same diagonal are *compatible* if no vertices other than $(0, 0)$ can reach both of them by a unit path. Since a path of critical nodes represents a parenthesization, all critical nodes are compatible. Also, D_n has at most $n - 1$ critical nodes and there is at least one path from $(0, 0)$ to $(1, n)$ that includes all critical nodes [11].

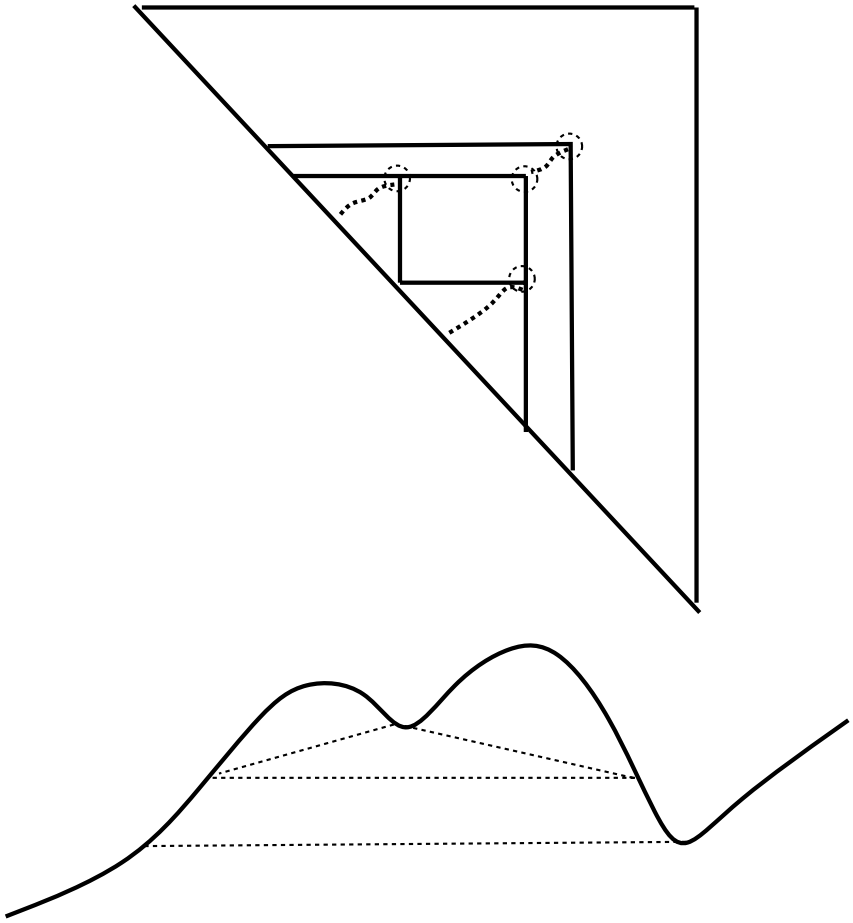


FIG. 2. *Two leaf subgraphs inside a band subgraph with critical nodes shown.*

2.2. Canonical subgraphs of D_n. In this subsection we investigate the interaction between subgraphs containing critical nodes.

All vertices and edges that can reach (i,t) by a unit path form the subgraph $D(i,t)$. Given $D(i,j)$ if the weight list $w_i,\ldots,w_{j+1}$ is monotonic, then $D(i,j)$ is monotonic. A *band canonical subgraph* $D^{(i,t)}_{(j,k)}$ is the subgraph containing the maximal unit edge-connected path of critical nodes beginning at critical node (j,k) and terminating at critical node (i,t) with the vertex set $\{(0,0)\}\cup(V[D(i,t)]-V[D(j+1,k-1)])$ and associated edges. A canonical subgraph of the form $D^{(i,t)}_{(j,j+1)}$ is a *leaf* canonical subgraph and is written $D^{(i,t)}$; it has the same nodes and edges as $D(i,t)$. The top of Figure 2 shows two leaf subgraphs nested inside of a band subgraph. Leaf and band subgraphs are the only two types of canonical subgraphs. Canonical subgraphs are easily distinguishable by the properties of their critical nodes shown in Theorem 2. From here on p denotes the path of critical nodes in band or leaf canonical subgraphs.

Given $D(i,u)$ with a monotone list of weights $w_i \le w_{i+1} \le \cdots \le w_{u+1}$, a shortest path from $(0,0)$ to (i,u) is the straight unit path $(0,0) \nearrow (i,i) \to (i,i+1) \to \cdots \to (i,u)$ that costs $w_i \sum_{j=i+1}^{u} w_j w_{j+1}$. On the other hand, if $D(i,t)$ has no

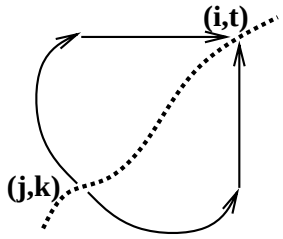


FIG. 3. *Two angular paths.*

critical nodes, then its associated weight list is monotonic. As in [28, 29, 11] let $\|w_i : w_k\| = \sum_{j=i}^{k-1} w_j w_{j+1}$, which is easily computable using differences of components of the parallel partial prefixes $\|w_1 : w_i\|$ for $2 \le i \le n+1$. This is useful since the unit path $(i,j) \to \cdots \to (i,k)$ costs $w_i\|w_{j+1} : w_{k+1}\| = w_i(\|w_1 : w_{k+1}\| - \|w_1 : w_{j+1}\|)$.

Suppose (j,k) and (i,t) are two critical nodes in a canonical graph such that from (j,k) we can reach (i,t) by a unit path, that is if $i \le j \le k \le t$, then the *angular* paths of (j,k) and (i,t) are, (see Figure 3)

$$(j,k) \Uparrow (i,k) \to \cdots \to (i,t) \text{ and } (j,k) \Longrightarrow (j,t) \uparrow \cdots \uparrow (i,t).$$

THEOREM 4 (see [11]). *In a canonical subgraph the shortest path between any two critical nodes that contains no other critical nodes is an angular path or edge.*

In addition, any shortest path not including critical nodes is a straight path of unit edges. Thus, any shortest path to a critical node that contains no other critical nodes is a straight path of unit edges [11].

Now a polylog-time algorithm for finding shortest paths to all critical nodes in $D^{(1,m)}$ graphs is given. This algorithm takes $O(\lg^2 m)$ time and uses $m^3/\lg m$ processors.

First compute the parallel partial prefixes $\|w_1 : w_i\|$ for $2 \le i \le m+1$. Find all critical nodes. Now, in constant time using m processors compute the costs of all of the unit paths to nodes in p. Next compute the cost of the $O(m^2)$ angular paths in constant time with m^2 processors. Finally, compute the shortest path to each node in p by treating every angular path as an edge and applying a parallel all-pairs shortest path algorithm.

2.3. Combining the canonical graphs for an efficient parallel algorithm. In this subsection we discuss a tree contraction algorithm that contracts the tree structure joining the canonical subgraphs to form a shortest path in D_n; see also [28, 29, 11].

In D_n a *canonical tree* joins all of the canonical subgraphs. A node in a canonical tree is critical node, say (i,j), and is written $\overline{(i,j)}$. Initially, for every leaf $D^{(i,j)}$ the critical node (i,j) is the tree leaf $\overline{(i,j)}$. Internal tree nodes are either isolated critical nodes or $\overline{(i,t)}$ and $\overline{(j,k)}$ in the band $D^{(i,t)}_{(j,k)}$. Tree edges are straight unit paths connecting tree nodes, and jumpers may reduce the cost of tree edges.

Given an instance of the MCOP with the weight list $l_1 = w_1, w_2, \ldots, w_{n+1}$, cyclically rotating it, getting l_2, and finding an optimal parenthesization for l_2 gives an optimal solution to the original instance of the MCOP with l_1, [28, 21]. So in the rest of this paper let w_1 denote the smallest weight in any weight list.

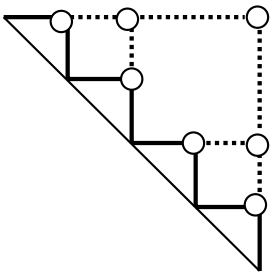


FIG. 4. *A tree of canonical graphs (the circles denote tree nodes).*

A result of Hu and Shing [28] leads directly to the next corollary.

COROLLARY 1 (Atomicity Corollary [11]). *Suppose a weight list* $w_1, \ldots, w_{n+1}$, *with the three smallest weights* w_1, w_{j+1}, *and* w_{k+1}, *is given such that* $1 < j < k-1$. *Then the critical nodes* $(1, j)$ *and* $(1, k)$ *are in a shortest path from* $(0, 0)$ *to* $(1, n)$ *in* D_n.

For this corollary to work it is central that if w_1, w_{j+1}, and w_{k+1} are the three smallest weights; then $j+1 > 2$ and $k > j+1$. This generally means that Corollary 1 cannot be applied in a canonical subgraph. For instance, take the leaf $D^{(1,m)}$ where we can assume $w_1 < w_{m+1} < w_i$ for $1 < i < m+1$. However, Corollary 1 can be used to break D_n into a tree of canonical graphs; see Figure 4.

If D_n has fewer than $n-1$ critical nodes, then D_n may have disconnected canonical trees and monotone subgraphs. But there is at least one path joining these subtrees, and at the same time we can discount the monotone subgraphs. There are several relationships canonical graphs may have; these follow directly from the relationships of critical nodes that are tree nodes.

The tree edge $\overline{(i,j)} \to \cdots \to \overline{(i,v)}$ along row i initially costs $w_i \| w_{j+1} : w_{v+1} \|$ where $w_i < w_{v+1} < w_{j+1}$ are the three smallest weights in $D(i, v)$. Let $\overline{p}$ denote a shortest path of critical nodes in $D(j+1, v)$ from $(j+1, v)$ back to $(0, 0)$. *Edge minimizing* the unit path along the ith row to the critical node $\overline{(i,v)}$ is done as follows. First let $L = w_i \| w_{i+1} : w_{v+1} \|$ and $W((i,k) \Longrightarrow (i,u)) = sp(k+1, u) + f(i, k, u)$, then compute

$$A[i,v] = \min_{\forall (k+1,u) \in V[\overline{p}]} \{L,\ w_i \| w_{i+1} : w_{v+1} \| - w_i \| w_{k+1} : w_{u+1} \| + W((i,k) \Longrightarrow (i,u))\}.$$

Since the three smallest weights in $D(i, v)$ are $w_i < w_{v+1} < w_{j+1}$, by Corollary 1 the cheapest cost to critical node (i, v) is now in $A[i, v]$.

THEOREM 5 (see [11]). *When edge minimizing a tree edge* $\overline{(i,j)} \to \cdots \to \overline{(i,v)}$ *in a canonical subgraph we only have to consider jumpers* $(i,k) \Longrightarrow (i,t)$ *such that* $(k+1, t) \in V[\overline{p}]$.

The critical node (i, u) in the band $D^{(i,u)}_{(j,k)}$ is the *front* critical node. In general, Theorem 5 holds when $\overline{p}$ is a shortest path through a band from the front critical node back to $(0, 0)$. Also, Theorem 5 holds for leaves in the canonical tree that, after raking, have become conglomerates of other leaves, bands, and isolated critical nodes. Here, jumpers derived from critical nodes in different subtrees are independent so we can minimize tree edges with them simultaneously.

2.4. Contracting a canonical tree. In this subsection we show how to contract a canonical tree efficiently in parallel.

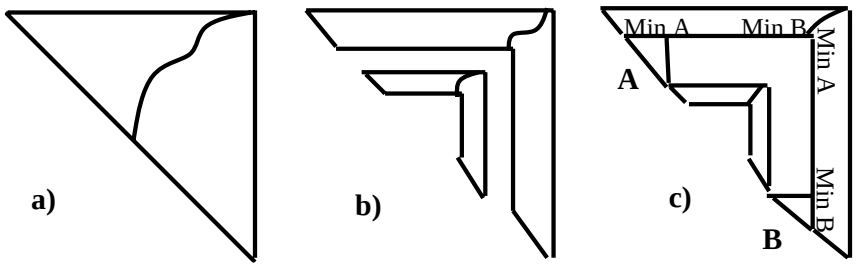


FIG. 5. *Bottlenecks* 1, 2, *and* 3 *for the* $n^3/\lg n$*-processor algorithm.*

Assume that all critical nodes (i, j) in tree leaves have the minimum cost back to $(0, 0)$ stored in $sp(i, j)$. Compute these values using an all-pairs shortest path parallel algorithm. There is an ordering of the leaves that prevents the simultaneous raking of two adjacent leaves. Given two neighboring leaves $D^{(i,j)}$ and $D^{(j+1,k)}$ with the three tree leaves $\overline{(i,j)}, \overline{(j+1,k)}$, and $\overline{(i,k)}$, assume $w_i < w_{k+1} < w_{j+1}$. Then leaf $\overline{(j+1,k)}$ must be raked, since $\overline{(i,j)}$ is in a shortest path from $(0, 0)$ to $\overline{(i,k)}$. Use the Euler tour technique [33] when the raking order is arbitrary.

Given two nested bands, assume $D^{(i,v)}_{(j,u)}$ is nested around $D^{(k,t)}_{(r,s)}$, that is, $j \leq k < t \leq u$. Without loss of generality, suppose any trees between $D^{(i,v)}_{(j,u)}$ and $D^{(k,t)}_{(r,s)}$ have been contracted. Then joining these bands costs $O(\lg^2 n)$ time with $n^3/\lg n$ processors. To achieve this, first, edge minimize all straight unit paths in $D^{(i,v)}_{(j,u)}$ with the shortest paths from critical nodes that are between $D^{(i,v)}_{(j,u)}$ and $D^{(k,t)}_{(r,s)}$ back to $(0, 0)$. Next, take all angular paths connecting these two bands and apply a parallel all-pairs shortest path algorithm merging the bands. Merging the bands $D^{(i,v)}_{(j,u)}$ and $D^{(j,u)}_{(r,s)}$ gives a shortest path from the front critical node (i, v) back to $(0, 0)$ through $D^{(i,v)}_{(r,s)}$. Incorporating this band merging with the tree contraction completes the polylog-time and $n^3/\lg n$-processor MCOP algorithm.

3. The structure of shortest paths in canonical subgraphs. In this section we give the $n^3/\lg n$-processor bottlenecks of the algorithm in section 2. In addition, we give a metric for measuring the relative contributions of angular paths to shortest paths and some theorems about shortest paths forward from critical nodes in canonical graphs. From this section on, we only address rows in the canonical graphs; the arguments for columns follow immediately.

3.1. The $n^3/\lg n$ processor bottlenecks. In this subsection we give the $n^3/\lg n$-processor bottlenecks of the algorithm sketched in section 2.

Three parts of the algorithm in section 2 use $n^3/\lg n$ processors. All other parts of this algorithm use a total of $n/\lg n$ processors and take $O(\lg n)$ time. The three bottlenecks are: finding shortest paths from all critical nodes in leaf graphs back to $(0, 0)$ (see Figure 5a); merging two bands (see Figure 5b); and merging two bands that have contracted canonical trees between them (see Figure 5c).

In Figure 5c, contracted trees **A** and **B** are used to edge minimize the unit paths marked by "**Min-A**" and "**Min-B**." Edge minimizing the unit paths in the outer band with the contracted trees gives an instance of the second bottleneck; see Figure 5b. Edge minimizing the unit paths in the outer band with the contracted trees costs $O(\lg n)$ time with $n^2/\lg n$ processors. In section 5 we will see how to perform such edge minimization in $O(\lg^2 n)$ time with $n/\lg n$ processors.

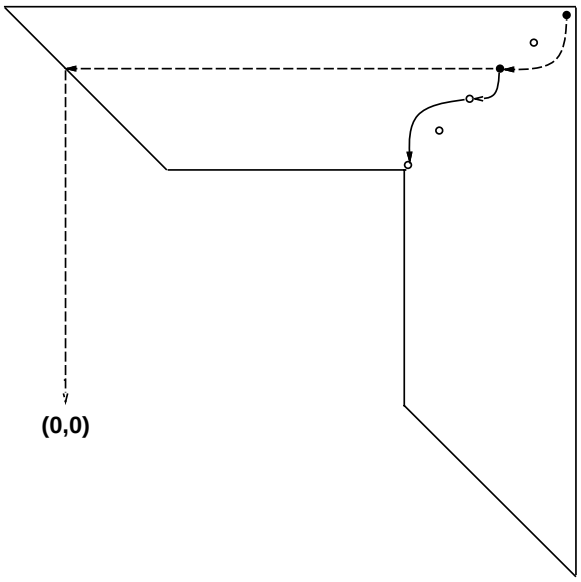


FIG. 6. *The dashed path is $\overline{p}$ and the two black nodes are supercritical nodes.*

Finding shortest paths back to $(0,0)$ from all critical nodes in a leaf graph, as in Figure 5a, will be done by breaking a leaf graph into nested bands. Therefore, finding efficient parallel methods of band merging and edge minimization will give an efficient parallel algorithm for the MCOP. So, the focus of the rest of the paper is finding efficient ways to get shortest paths from all critical nodes back to $(0,0)$ by edge minimization in leaf subgraphs partitioned as bands.

Given the band $D^{(i,v)}_{(j,t)}$, let $\overline{p}^{(i,v)}_{(j,t)}$ denote a shortest path from (i,v) back to $(0,0)$ totally contained in $D^{(i,v)}_{(j,t)}$; see Figure 6. When there is no ambiguity, $\overline{p}^{(i,v)}_{(j,t)}$ will be written as $\overline{p}$. Given a band $D^{(i,v)}_{(j,t)}$, whenever $\overline{p} = \overline{p}^{(i,v)}_{(j,t)}$ starts from the front critical node of the band it is in, the nodes $V[\overline{p}]$ are *super*critical nodes. Considering the minimal path back from the front critical node in Figure 6, we can see that only the two black critical nodes are supercritical nodes. Supercritical nodes of any band are all critical nodes in some minimal path back form the front critical node in the band back to $(0,0)$. Any two supercritical nodes in $\overline{p}$ are connected by supercritical nodes interspersed with the angular paths shown by Theorem 4.

When a canonical tree of D_n is totally contracted, then the final path $\overline{p}$ from $(1,n)$ back to $(0,0)$ gives the optimal order to multiply the set of n matrices. In addition, the cost of $\overline{p}$ is the minimal cost of multiplying the given chain of n matrices.

3.2. A metric for finding minimal cost angular paths. In this subsection we give a metric for finding minimal cost angular paths by using the equivalence of angular paths and jumpers along unit paths. This equivalence comes directly from Theorem 1.

When merging two bands, a unit path has at most one jumper minimizing it, since all the relevant jumpers are nested. These jumpers get their *sp* values from supercritical nodes of the inner band.

The influence of an angular edge can be taken as a jumper in a straight unit path by Theorem 1. In the case of Figure 5c, notice that any unit edge minimization using

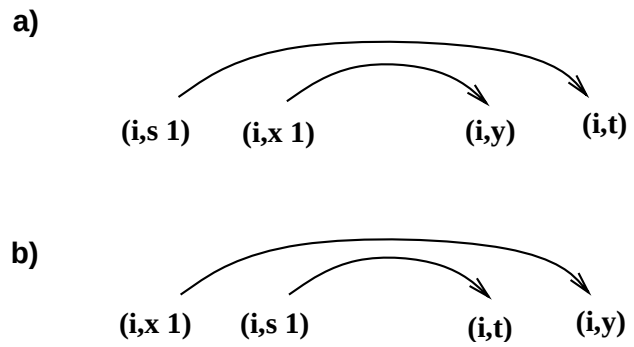


FIG. 7. *Two different nestings of two jumpers.*

sp values from **A** or **B** is independent of unit edge minimization using *sp* values from the inner band. Therefore, measuring the potential contribution of angular edges to shortest paths is done by measuring the potential contribution of jumpers to shortest paths along straight unit paths.

Take a node $(s,t) \in V[\overline{p}]$, where $sp(s,t)$ is the cost of a shortest path back to $(0,0)$ with respect to a band; then in row i we want to compare the cost of the jumper $(i,s-1) \Longrightarrow (i,t)$ with the cost of the associated unit path $(i,s-1) \rightarrow \cdots \rightarrow (i,t)$.

Given $(s,t) \in V[\overline{p}]$, take row i above p with the jumper $(i,s-1) \Longrightarrow (i,t)$; define

$$\Delta_i(s,t) = w_i \| w_s : w_{t+1} \| - [\ sp(s,t) + f(i,s-1,t)\ ].$$

If $\Delta_i(s,t) > 0$, then the jumper $(i,s-1) \Longrightarrow (i,t)$ provides a *cheaper* path along row i than the unit path $(i,s-1) \rightarrow \cdots \rightarrow (i,t)$. In particular, take both $(s,t) \in V[\overline{p}]$ and $(x,y) \in V[\overline{p}]$, and the two possible jumper nestings of Figure 7.

Considering the nesting of the jumpers in Figure 7a, if $\Delta_i(s,t) > \Delta_i(x,y) > 0$, then the jumper $(i,s-1) \Longrightarrow (i,t)$ "saves more" than the jumper $(i,x-1) \Longrightarrow (i,y)$ along row i because $(i,s-1) \Longrightarrow (i,t)$ doesn't have to deal with the paths $(i,s-1) \rightarrow \cdots \rightarrow (i,x-1)$ and $(i,y) \rightarrow \cdots \rightarrow (i,t)$ and $\Delta_i(s,t) > \Delta_i(x,y) > 0$. Similarly, for the nesting of the jumpers in Figure 7b, if $\Delta_i(x,y) > \Delta_i(s,t) > 0$, then the jumper $(i,x-1) \Longrightarrow (i,y)$ "saves more" than the jumper $(i,s-1) \Longrightarrow (i,t)$ along row i. Notice that considering the jumpers in Figure 7**b**, if $\Delta_i(s,t) > \Delta_i(x,y) > 0$, then the jumper $(i,s-1) \Longrightarrow (i,t)$ may or may not make row i cheaper than the jumper $(i,x-1) \Longrightarrow (i,y)$. On the other hand, in Figure 7b, if $(i,s-1) \Longrightarrow (i,t)$ makes row i cheaper than the jumper $(i,x-1) \Longrightarrow (i,y)$ does, then $\Delta_i(s,t) > \Delta_i(x,y) > 0$.

In $D^{(1,m)}$, if $(s,t) \in V[p]$, then above the path of critical nodes p the function $\Delta_i(s,t)$ is defined for all rows i such that $s > i \geq 1$.

Notice that edge minimizing a unit path is only half the game, for we also must consider the shortest paths forward.

Figure 8 is for the next theorem; also see [28].

THEOREM 6. *Let $D^{(i,v)}_{(r,s)}$ be a leaf graph and let (j,u) and (k,t) be any two critical nodes in $D^{(i,v)}_{(r,s)}$ such that there is a unit path from (k,t) to (j,u). Then a shortest path from (j,u) to (i,v) costs less than a shortest path from (k,t) to (i,v).*

A proof follows inductively by shadowing trivial angular paths without any jumpers, then showing that any shortest path from (k,t) forward to (i,v) can be

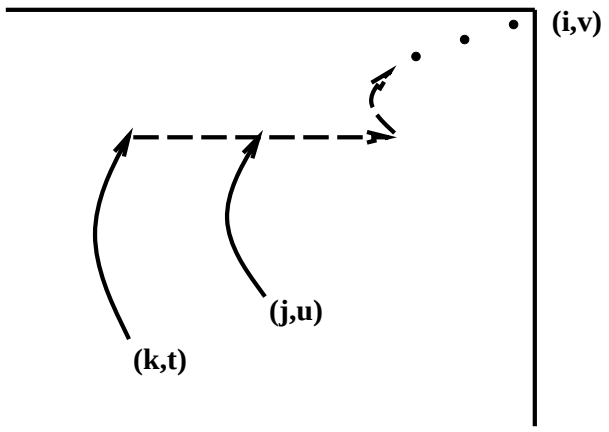


FIG. 8. *(j,u) shadowing (k,t)'s shortest path forward.*

"shadowed" by a shorter path from (j,u) forward to (i,v). While in the process we have taken into account the f values. Naturally, Theorem 6 also holds for shortest paths forward in leaf graphs.

The next theorem will also be useful.

THEOREM 7. *Let $(i,s-1) \Longrightarrow (i,t)$ be a shortest path forward. Suppose that the next band merging the value of $sp(s,t)$ decreases due to an edge minimization of row s or a lower row. Then $(i,s-1) \Longrightarrow (i,t)$ is still in a shortest path forward.*

A proof of this theorem follows directly from the basic notions of shortest paths. In particular, if the shortest path forward from the critical node (s,t) goes through $(s,t) \Uparrow (i,t)$, then making the path to (s,t) shorter will not affect the jumper $(s,t) \Uparrow (i,t)$ or the path from (i,t) to the front node of the present band.

4. A polylog-time and $n^2/\lg n$-processor MCOP algorithm. In this section we give an $O(\lg^2 n)$-time and $n^2/\lg n$-processor algorithm for the MCOP. This algorithm works by using a key induction invariant that allows recursive doubling techniques to break through the bottlenecks given in the last section.

The basic idea of the algorithm is as follows. All critical nodes know their shortest paths to the front of the present bands they are in. Only supercritical nodes have their shortest paths back to $(0,0)$ through their present bands. When merging two bands, by Theorem 5, we only have to consider shortest paths from supercritical nodes in the inner band to any critical node in the outer band. Therefore, all critical nodes must maintain a shortest path to the front of the band they are in. At the same time, all supercritical nodes must maintain a shortest path backwards to $(0,0)$ through the band they are in. Much of this section supplies the details and correctness of this algorithm.

Each critical node in D_n has two pointers called *front-ptr* and *back-ptr* that represent angular edges. *Back-ptr*s are only used by supercritical nodes. With each *front-ptr* there are two values, *cost-of-front-ptr* and *cost-to-front*; and with each *back-ptr* there is one value, *cost-to-back*. *Cost-of-front-ptr* is the cost of the angular edge going forward to the front critical node in the present band, where the value of *cost-to-front* is the entire cost to the front critical node of the present band containing *front-ptr*. Similarly, the value of *cost-to-back* is the cost from the supercritical node at hand back to $(0,0)$ through the present band. Initially, these pointers connect critical nodes and tree edges in the canonical tree.

1. All critical nodes in both bands have their *front-ptr*s in trees of shortest paths that eventually go to supercritical nodes. The supercritical nodes have their *front-ptr*s form a linked list that goes to the front (super)critical nodes of their respective bands.

2. In the two bands both shortest paths back to $(0,0)$ of supercritical nodes are known. These shortest paths of supercritical nodes are made of linked lists of *back-ptrs* from the front (super)critical nodes of each band back through their respective bands to $(0,0)$.

FIG. 9. *Inductive invariant for band merging.*

Let $D^{(i,v)}_{(j,t)}$ and $D^{(j,t)}_{(k,s)}$ be nested bands with paths of critical nodes labeled $p^{(i,v)}_{(j,t)}$ and $p^{(j,t)}_{(k,s)}$, respectively. Note (i,v) and (j,t) are the front critical nodes of these bands. As before, $\overline{p}^{(i,v)}_{(j,t)}$ and $\overline{p}^{(j,t)}_{(k,s)}$ are shortest paths from the front critical nodes back to $(0,0)$ through the bands $D^{(i,v)}_{(j,t)}$ and $D^{(j,t)}_{(k,s)}$, respectively. Let $\overline{p}^{(i,v)}_{(j,t)}$ and $\overline{p}^{(j,t)}_{(k,s)}$ be made by two linked lists of *back-ptrs* along supercritical nodes back to $(0,0)$ in their bands. It turns out that the shortest paths forward form all critical nodes in each of these bands and are made up of linked lists of trees of *front-ptr*s. We will see that this linked list of trees of *front-ptr*s is interconnected through the supercritical nodes as in Figure 10.

Figure 9 gives the induction invariant for merging two bands.

Figure 10 gives an example of the data structures for maintaining the inductive invariant. In this figure only critical nodes are shown and the supercritical nodes are black. The solid arrows are *front-ptr*s and the dashed arrows are *back-ptr*s.

Now, say (s,t) is a critical node but not a supercritical node, that is $(s,t) \in V[p]$ and $(s,t) \notin V[\overline{p}]$. There is a unique angular edge $(x,y) \Uparrow (r,y) \rightarrow \cdots \rightarrow (r,u)$ in $\overline{p}$ that "goes around" (s,t); see Figure 11. If we consider all rows above p in a given canonical graph, then $w_i < w_r$ implies that row i is "above" row r as in Figure 11. From here on we focus on finding shortest paths above the path p of critical nodes. The symmetric case of shortest paths below the path p of critical nodes follows.

Once we edge minimize all unit paths in $D^{(i,v)}_{(j,k)}$ with jumpers that get their *sp* values from supercritical nodes in $D^{(j,k)}_{(s,t)}$, then we can find the shortest path from (i,v) back to $(0,0)$ through $D^{(i,v)}_{(s,t)}$. First, take one processor at each critical node in $D^{(i,v)}_{(j,k)}$ that sums the cost of the path back to $(0,0)$, possibly through an edge-minimized unit path with the cost of its shortest path forward. Next, find the minimum of all of these sums, giving the shortest path from (i,v) back to $(0,0)$ through $D^{(j,k)}_{(s,t)}$.

The basic intuition for the next lemma is that, if the shorter of two nested jumpers edge minimizes a unit path r, then any unit path above r with both of these jumpers is not minimized by the longer jumper; see Figure 12. For the next lemma, assume there is a unit path of critical nodes from (x,y) to (s,t) to (r,u) as in Figure 11.

LEMMA 1. *Let (s,t) be a critical node between the* super*critical node (x,y) and the critical node (r,u) and suppose that $i < r < s < x$ and row i is above row r. That is, $w_i < w_r$, where rows i and r are above p. Then*

$$\textbf{if } \Delta_r(x,y) \geq \Delta_r(s,t), \textbf{ then } \Delta_i(x,y) \geq \Delta_i(s,t).$$

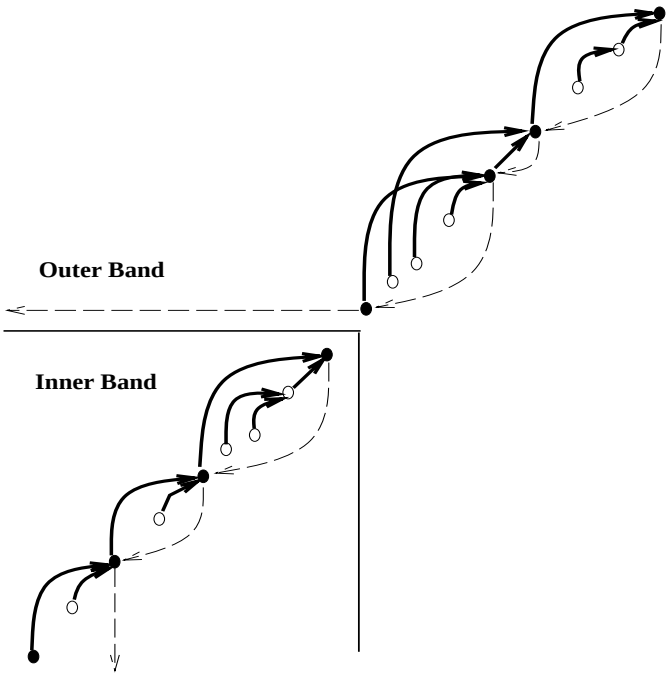


FIG. 10. *Solid arrows: forward linked lists of trees; dashed arrows: backward linked lists* $\overline{p}$.

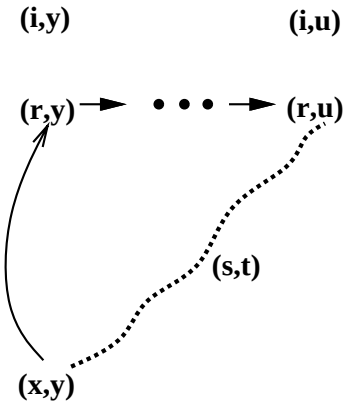


FIG. 11. $(s,t) \notin V[\overline{p}]$ *and the angular edge* $(x,y) \Uparrow (r,y) \rightarrow \cdots \rightarrow (r,u)$.

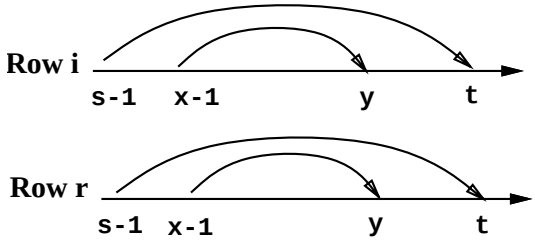


FIG. 12. *Two jumpers in different rows.*

Proof. Suppose $\Delta_r(x,y) \geq \Delta_r(s,t)$. This means

$$w_r\|w_x : w_{y+1}\| - [sp(x,y) + f(r,x-1,y)] \geq w_r\|w_s : w_{t+1}\| - [sp(s,t) + f(r,s-1,t)].$$

Using some algebra we obtain the following (where $\|w_i : w_i\| = 0$):

$$w_r[\ \|w_s : w_x\| + \|w_{y+1} : w_{t+1}\|\ ] < sp(s,t) - sp(x,y) + w_r(w_s w_{t+1} - w_x w_{y+1}).$$

Moreover, $sp(s,t) - sp(x,y)$ is always positive because $(r,x-1) \Longrightarrow (r,y)$ is nested inside of $(r,s-1) \Longrightarrow (r,t)$ and $f(r,s-1,t) < f(r,x-1,y)$. Therefore, if $sp(x,y) > sp(s,t)$, then a shortest path $\overline{p}$ would go through (s,t) to (r,u) and *not* over (s,t). In particular, if $sp(x,y) > sp(s,t)$, then since $f(r,x-1,t) > f(r,s-1,t)$, it must be the case that $sp(x,y) + f(r,x-1,t) > sp(s,t) + f(r,s-1,t)$. Therefore, row r would have been edge minimized by jumper $(r,s-1) \Longrightarrow (r,t)$ and *not* by $(r,x-1) \Longrightarrow (r,y)$; see Figure 12.

In addition, $w_s w_{t+1} - w_x w_{y+1} < 0$, since both (x,y) and (s,t) are critical nodes where $s \leq x < y \leq t$. So it must be that $w_x w_{y+1} - w_s w_{t+1} > 0$. Therefore, since

$$w_r[\ \|w_s : w_x\| + \|w_{y+1} : w_{t+1}\| + w_x w_{y+1} - w_s w_{t+1}\ ] < sp(s,t) - sp(x,y)$$

holds, and because $w_i < w_r$ and the term $sp(s,t) - sp(x,y)$ is independent of i and r, then $\Delta_i(x,y) \geq \Delta_i(s,t)$ follows. □

The next theorem follows from Lemma 1.

THEOREM 8. *Let (s,t) be a critical node between the* super*critical node (x,y) and the critical node (r,u). Suppose $i < r < s < x$ and row i is above row r, that is, $w_i < w_r$, where rows i and r are above p. Then*

if $(r,x-1) \Longrightarrow (r,y)$ makes row r cheaper than $(r,s-1) \Longrightarrow (r,t)$ does,
then $(i,x-1) \Longrightarrow (i,y)$ makes row i cheaper than $(i,s-1) \Longrightarrow (i,t)$ does.

A proof follows from Lemma 1 and by the fact that the rows

$$(i,s-1) \rightarrow \cdots \rightarrow (i,x-1) \text{ and } (i,y) \rightarrow \cdots \rightarrow (i,t)$$

are cheaper than the rows

$$(r,s-1) \rightarrow \cdots \rightarrow (r,x-1) \text{ and } (r,y) \rightarrow \cdots \rightarrow (r,t),$$

and the change in f values between $(r,x-1) \Longrightarrow (r,y)$ and $(i,x-1) \Longrightarrow (i,y)$ is greater than the change of f values between $(r,s-1) \Longrightarrow (r,t)$ and $(i,s-1) \Longrightarrow (i,t)$. That is,

$$f(r,x-1,y) - f(i,x-1,y) > f(r,s-1,t) - f(i,s-1,t),$$

since w_x and w_{y+1} are both bigger than w_s and w_{t+1}. In addition, $w_r > w_i$; therefore

$$(w_r - w_i)[w_x w_{y+1} - w_s w_{t+1}] > 0.$$

Consider two nested bands with paths of critical nodes p_i for the inner band and p_o for the outer band, where $\overline{p_i}$ and $\overline{p_o}$ are shortest paths from the front critical nodes back to $(0,0)$ in each of these bands. Now, suppose (s,t) is between (x,y) and (r,u) and $(s,t) \in V[\overline{p_i}]$. If $(x,y) \in V[\overline{p_i}]$ and $(r,u) \in V[p_o]$, then Lemma 1 and Theorem 8

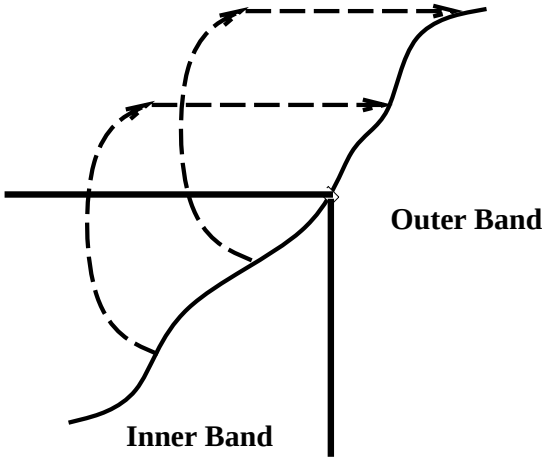


FIG. 13. *Conflicting angular paths* between *two bands being merged.*

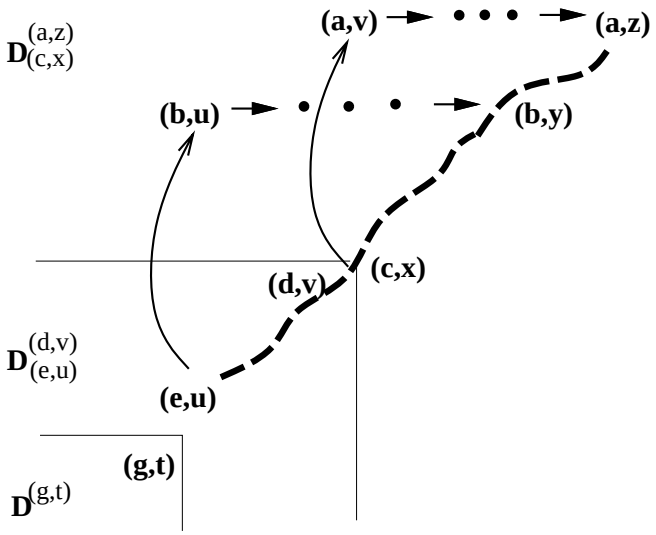


FIG. 14. *The bands* $D^{(a,z)}_{(c,x)}$, $D^{(d,v)}_{(e,u)}$ *and the leaf* $D^{(g,t)}$.

also hold. This is because $sp(s,t) - sp(x,y)$ is positive by an argument similar to that in the proof of Lemma 1.

Two angular edges above p, say $(x,y) \Uparrow (r,y) \to \cdots \to (r,u)$ and $(i,j) \Uparrow (s,j) \to \cdots \to (s,t)$, are *compatible* if they don't cross each other. Compatibility also holds for angular paths below p. Theorem 9 shows that *when merging* two bands and computing shortest paths forward, only compatible angular edges need to be considered. Figure 13 shows two conflicting angular paths.

Take the canonical graphs $D^{(a,z)}_{(c,x)}$, $D^{(d,v)}_{(e,u)}$ and $D^{(g,t)}$, where $D^{(g,t)}$ is nested inside of $D^{(d,v)}_{(e,u)}$ which is, in turn, inside of $D^{(a,z)}_{(c,x)}$; see Figure 14. Furthermore, assume that $D^{(a,z)}_{(c,x)}$ and $D^{(d,v)}_{(e,u)}$ are to be merged together. Then, in the next recursive doubling step, the new band $D^{(a,z)}_{(e,u)}$ will be merged with the leaf $D^{(g,t)}$. We can assume $D^{(g,t)}$ is a leaf or a band.

The next theorem assumes we have found a shortest path from supercritical nodes in $D^{(d,v)}_{(e,u)}$, through critical nodes in the outer band $D^{(a,z)}_{(c,x)}$; see Figure 14. We know $(d,v) \in V[\overline{p}^{(d,v)}_{(e,u)}]$ and, without loss of generality, we can assume $(e,u) \Uparrow (d,u) \rightarrow \cdots \rightarrow (d,v)$ is $\overline{p}^{(d,v)}_{(e,u)}$. Now, suppose the angular edge $(e,u) \Uparrow (b,u) \rightarrow \cdots \rightarrow (b,y)$ is in $\overline{p}^{(a,z)}_{(e,u)}$, where $\overline{p}^{(a,z)}_{(e,u)}$ is the minimal path from (a,z) back to $(0,0)$ through $D^{(a,z)}_{(e,u)}$.

THEOREM 9 (Main Theorem). *In merging two nested bands computing shortest paths forward from supercritical nodes of the inner band, we only need to consider compatibly nested angular edges.*

Proof. The proof is by contradiction. Suppose $(e,u) \Uparrow (b,u) \rightarrow \cdots \rightarrow (b,y)$ is in $\overline{p}^{(a,z)}_{(e,u)}$, that is, the angular edge $(e,u) \Uparrow (b,u) \rightarrow \cdots \rightarrow (b,y)$ is in a shortest path from (a,z) back to $(0,0)$ through $D^{(a,z)}_{(e,u)}$; see Figure 14. Suppose (d,v) is in the band $D^{(d,v)}_{(e,u)}$. Therefore (d,v) is between (e,u) and (a,z) in $D^{(a,z)}_{(e,u)}$. Now, when merging $D^{(g,t)}$ with $D^{(a,z)}_{(e,u)}$ we will show that a shortest path forward to (a,z) that goes through (d,v) must go through a critical node in row b or a critical node in a row below b.

Now, for the sake of a contradiction, assume otherwise. Suppose after merging $D^{(a,z)}_{(c,x)}$ with $D^{(d,v)}_{(e,u)}$ there is some shortest path from $(0,0)$ through (d,v) to (a,z). This shortest path travels through an angular path connecting the bands $D^{(a,z)}_{(c,x)}$ and $D^{(d,v)}_{(e,u)}$ and this angular path is conflicting with the angular path $(e,u) \Uparrow (b,u) \rightarrow \cdots \rightarrow (b,y)$. Say, without loss of generality, this conflicting angular path is $(d,v) \Uparrow (a,v) \rightarrow \cdots \rightarrow (a,z)$; see Figure 14. That is, we have conflicting angular paths since the shortest path from (d,v) forward goes through an angular path that terminates above row b, and the shortest path forward from (e,u) goes through an angular path that terminates in row b. But notice *in* $D^{(a,z)}_{(e,u)}$ that the shortest path from (a,z) back to $(0,0)$ still goes through the angular edge $(e,u) \Uparrow (b,u) \rightarrow \cdots \rightarrow (b,y)$.

In $D^{(a,z)}_{(e,u)}$ the angular edge $(d,v) \Uparrow (a,v) \rightarrow \cdots \rightarrow (a,z)$ can't be the shortest path forward from (d,v).

By Theorem 1, the shortest path to (b,y) through the angular edge

$$(e,u) \Uparrow (b,u) \rightarrow \cdots \rightarrow (b,y)$$

is equivalent to the path

$$(b,b) \rightarrow \cdots \rightarrow (b,e-1) \Longrightarrow (b,u) \rightarrow \cdots \rightarrow (b,y).$$

Moreover, since $\overline{p}^{(a,z)}_{(e,u)}$ goes through $(e,u) \Uparrow (b,u) \rightarrow \cdots \rightarrow (b,y)$, the jumper $(b,e-1) \Longrightarrow (b,u)$ edge minimizes row b. Thus, the jumper $(b,d-1) \Longrightarrow (b,v)$ saves at most as much as $(b,e-1) \Longrightarrow (b,u)$, and $(b,d-1) \Longrightarrow (b,v)$ is nested around $(b,e-1) \Longrightarrow (b,u)$. Thus,

$$\Delta_b(e,u) \geq \Delta_b(d,v).$$

Also, by Theorem 1, the shortest path from (d,v) to (a,z) that goes through the angular edge

$$(d,v) \Uparrow (a,v) \rightarrow \cdots \rightarrow (a,z)$$

is equivalent to the path

$$(a,a) \rightarrow \cdots \rightarrow (a,d-1) \Longrightarrow (a,v) \rightarrow \cdots \rightarrow (a,z).$$

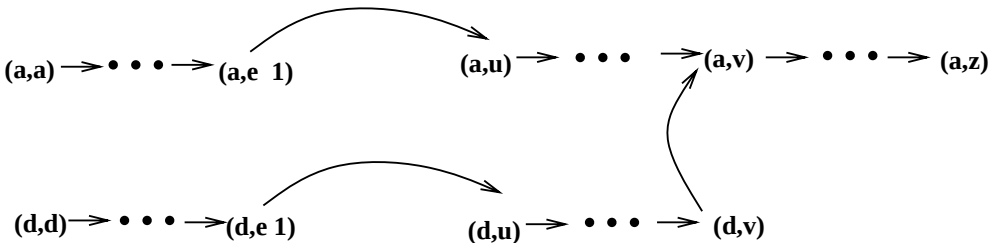


FIG. 15. *The two paths $\mathcal{A}$ and $\mathcal{D}$.*

But, consider the path

$$(a,a) \to \cdots \to (a,e-1) \Longrightarrow (a,u) \to \cdots \to (a,z),$$

and we know that the jumper $(a,e-1) \Longrightarrow (a,u)$ is nested inside of $(a,d-1) \Longrightarrow (a,v)$. In this case, it is possible that $d = e$ or $u = v$, but not both, since (d,v) is between (e,u) and (b,y) and $a < b$ and $w_a < w_b$, where row a is above row b and they are both above p. Furthermore, since the appropriate Δ values are defined, the following holds by Lemma 1:

$$\textbf{if } \Delta_b(e,u) \geq \Delta_b(d,v), \textbf{ then } \Delta_a(e,u) \geq \Delta_a(d,v).$$

Therefore,

$$\Delta_a(e,u) \geq \Delta_a(d,v),$$

which means the jumper $(a,e-1) \Longrightarrow (a,u)$ saves at least as much as the jumper $(a,d-1) \Longrightarrow (a,v)$ in a path to (a,z).

By Theorem 8, since $(b,e-1) \Longrightarrow (b,u)$ edge minimizes row b, and $\Delta_a(e,u) \geq \Delta_a(d,v)$, the jumper $(a,e-1) \Longrightarrow (a,u)$ saves more in row a than $(a,d-1) \Longrightarrow (a,v)$.

Now, take the two paths

$$\mathcal{A} = (a,a) \to \cdots \to (a,e-1) \Longrightarrow (a,u) \to \cdots \to (a,v),$$
$$\mathcal{D} = (d,d) \to \cdots \to (d,e-1) \Longrightarrow (d,u) \to \cdots \to (d,v) \Uparrow (a,v)$$

as in Figure 15.

$\mathcal{A}$ is cheaper than $\mathcal{D}$ going from (a,z) back to $(0,0)$ in $D^{(a,z)}_{(e,u)}$ by Theorem 8. Now, if $(d,v) \Uparrow (a,v)$ is in a shortest path forward from (d,v), then the shortest path forward from (e,u) must be through the angular path $(e,u) \Uparrow (a,u) \to \cdots \to (a,z)$ and not the angular path $(e,u) \Uparrow (b,u) \to \cdots \to (b,y)$, which is a contradiction. This follows by applying Theorem 8 to the jumpers $(b,d-1) \Longrightarrow (b,v)$ and $(b,e-1) \Longrightarrow (b,u)$ in row b and then up to row a, since (b,y) is between (d,v) and (a,z).

Now, suppose $D^{(g,t)}$ is merged with the outer band $D^{(a,z)}_{(e,u)}$. Then, none of the angular paths connecting supercritical nodes in $D^{(g,t)}$ with paths forward $D^{(a,z)}_{(e,u)}$ change. This case is a straightforward application of the proof above and Theorem 7. ∎

It is important to note that Theorem 9 shows that only angular paths starting from supercritical nodes in the same path back to $(0,0)$ are compatible. Theorem 9 doesn't say that all angular paths are always compatible.

Suppose that there is some angular path from a supercritical node in the inner band, say (s,t), to the outer band that is in a shortest path from the front node

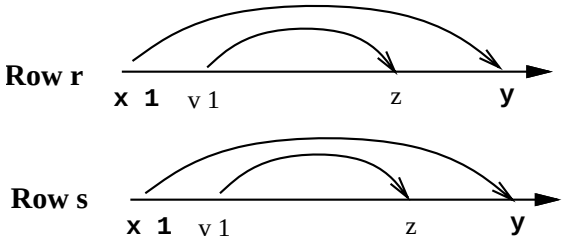


FIG. 16. *Two jumpers in different rows.*

of the outer band back to $(0,0)$. Then all supercritical nodes from (s,t) back to $(0,0)$ have their shortest paths forward through the angular path starting at (s,t). On the other hand, by Theorem 9 all supercritical nodes after (s,t) up to the front supercritical node of the inner band have their shortest paths through nested angular paths connecting the inner and outer bands. In fact, we can inductively apply this argument together with Theorem 6 giving the following corollary.

COROLLARY 2. *Consider the nested angular paths connecting two bands that are shortest paths forward from the different supercritical nodes of the inner band. Then, listing the path containing the outermost such angular path to the path containing the innermost such angular path gives more and more costly paths forward.*

The next lemma assumes we are merging two nested bands to find a shortest path from the front critical node of the outer band back to $(0,0)$.

LEMMA 2. *Let (s,t) be a critical node and let the ith and rth rows above p be such that $i < r < s$ and $w_i < w_r$. Then $\Delta_i(s,t) < \Delta_r(s,t)$.*

Proof. The function $\Delta_i(s,t)$ measures the potential minimizing effect of $(i, s-1) \Longrightarrow (i,t)$ on the path $(i,i) \rightarrow \cdots \rightarrow (i,u)$, where $(i,u) \in V[p]$ and $i < s < t \leq u$. The cost of the jumper $(i, s-1) \Longrightarrow (i,t)$ is $sp(s,t) + f(i, s-1, t)$. Therefore, the difference $\Delta_{i+1}(s,t) - \Delta_i(s,t)$ is

$$(w_{i+1} - w_i)[\ \|w_s : w_{t+1}\| - w_s w_{t+1}\ ],$$

where $w_{i+1} > w_i$. Since the expression $\|w_s : w_{t+1}\| - w_s w_{t+1}$ is independent of the difference of weights w_i and w_{i+1} and $\|w_s : w_{t+1}\| - w_s w_{t+1} > 0$, because $(s,t) \in V[p]$. Also, when $s = t-1$ we have

$$\|w_s : w_{t+1}\| = w_s w_{s+1} + w_{s+1} w_{t+1}.$$

In addition, since $(s,t) \in V[p]$, it must be that $\max\{w_s, w_{t+1}\} < w_u$, for $s < u \leq t$. Thus $\max\{w_s, w_{t+1}\} < w_{s+1}$. Therefore,

$$w_s w_{s+1} + w_{s+1} w_{t+1} > w_s w_{t+1}$$

and the proof follows inductively. □

The proof of the next lemma is similar to that of Lemma 1. The basic intuition here is that, if the longer of two nested jumpers edge minimizes a unit path r, then any unit path below r, with both of these jumpers, is not minimized by the shorter jumper; see Figure 16.

This next lemma only considers supercritical nodes since we are interested in merging two nested bands. For the next lemma assume there is a unit path of critical nodes from (v,z) to (x,y).

LEMMA 3. *Let* (v,z) *and* (x,y) *be two* super*critical nodes, where* $r<s<x<v$, *and assume* $w_r<w_s$ *such that rows* s *and* r *are above* p. *Then*

$$\textbf{if } \Delta_r(x,y)\geq\Delta_r(v,z), \textbf{ then } \Delta_s(x,y)\geq\Delta_s(v,z).$$

Proof. Assume $\Delta_r(x,y)\geq\Delta_r(v,z)$. Then

$$\begin{aligned} &w_r\|w_x:w_{y+1}\|-[\ sp(x,y)+f(r,x-1,y)\ ]\\ &\quad\geq w_r\|w_v:w_{z+1}\|-[\ sp(v,z)+f(r,v-1,z)\ ]. \end{aligned}$$

By Lemma 2 and, since each of these jumpers is of length at least 2, we know that $w_rw_vw_{z+1}<w_r\|w_v:w_{z+1}\|$ and $w_rw_xw_{y+1}<w_r\|w_x:w_{y+1}\|$. In addition, since $w_r<w_s$, we know that $f(r,x-1,y)<f(r,v-1,z)$ and $w_r\|w_x:w_{y+1}\|>w_r\|w_v:w_{z+1}\|$. Furthermore, the same holds in row s. Therefore, it must be that $\Delta_s(x,y)\geq\Delta_s(v,z)$. □

THEOREM 10. *Suppose we are given two* super*critical nodes* (v,z) *and* (x,y), *where* $r<s<x<v$, *and* $w_r<w_s$ *such that rows* s *and* r *are above* p. *Then*

if $(r,x-1)\Longrightarrow(r,y)$ *makes row* r *cheaper than* $(r,v-1)\Longrightarrow(r,z)$ *does,*

then $(s,x-1)\Longrightarrow(s,y)$ *makes row* s *cheaper than* $(s,v-1)\Longrightarrow(s,z)$ *does.*

A proof of this theorem follows from Lemma 3 and the fact that the change of the f values between $(r,v-1)\Longrightarrow(r,z)$ and $(s,v-1)\Longrightarrow(s,z)$ increases faster than the change in the f values between $(r,x-1)\Longrightarrow(r,y)$ and $(s,x-1)\Longrightarrow(s,y)$.

While merging $D^{(i,v)}_{(j,t)}$ and $D^{(j,t)}_{(k,s)}$ to form $\overline{p}^{(i,v)}_{(k,s)}$, the next lemma shows that we only need shortest path values backwards to $(0,0)$ from supercritical nodes and we don't need shortest path values backwards to $(0,0)$ from any other critical nodes. Hence, the *back-ptr*s will form a linked list between supercritical nodes backwards eventually to $(0,0)$, and we can compute the *cost-to-back* weights using a parallel pointer jumping partial prefix computation.

LEMMA 4. *Suppose we are given* $\overline{p}^{(i,v)}_{(j,t)}$ *and* $\overline{p}^{(j,t)}_{(k,s)}$ *in* $D^{(i,v)}_{(j,t)}$ *and* $D^{(j,t)}_{(k,s)}$, *respectively. Consider critical nodes in the outer band, say* $(u,z)\in V[p^{(i,v)}_{(j,t)}]$ *and* $(u,z)\notin V[\overline{p}^{(i,v)}_{(k,s)}]$; *then we don't need shortest paths* back *to* $(0,0)$.

Proof. Consider the angular path $(x,y)\Uparrow(q,y)\rightarrow\cdots\rightarrow(u,z)$ between $D^{(j,t)}_{(k,s)}$ and $D^{(i,v)}_{(j,t)}$. That is, $(x,y)\in V[\overline{p}^{(j,t)}_{(k,s)}]$ and $(u,z)\in V[p^{(i,v)}_{(j,t)}]$. But, suppose $(u,z)\notin V[\overline{p}^{(i,v)}_{(k,s)}]$ is the case. Notice that (u,z) may be a supercritical node in $\overline{p}^{(i,v)}_{(j,t)}$.

Consider the following cases.

Case i: Suppose $D^{(i,v)}_{(k,s)}$ is merged with another band nested around it.

Then, since (u,z) is not in $V[\overline{p}^{(i,v)}_{(k,s)}]$, by Theorem 5 we do not have to consider any angular paths starting from (u,z) going forward to critical nodes in the band nested around $D^{(i,v)}_{(k,s)}$.

Case ii: Suppose $D^{(i,v)}_{(k,s)}$ is merged with a smaller band inside $D(k,s)$.

Node (u,z) could be the terminal node of an incoming angular path contributing to a shortest path forward for *some* supercritical node in $D(k,s)$. In this case (u,z) needs to have a shortest path from (u,z) forward. Of course, in this case (u,z) could become a supercritical node and would have a minimal path back to $(0,0)$. On the other hand, since the critical node (u,z) is not a *super*critical node it has no need of a shortest path back to $(0,0)$. □

We want to find a shortest path forward for every critical node, since some angular path from some future inner band may terminate at any critical node. Therefore, after finding each supercritical node's minimal cost to the front critical node of the outer band, then compute a tree partial prefix sum from the critical nodes to the supercritical nodes. This lets all critical nodes know their shortest paths to the front of the outer band.

Suppose, through recursive doubling, we generate the band $D_{(j,t)}^{(i,v)}$ and the shortest path $\overline{p}_{(j,t)}^{(i,v)}$ from (i, v) back to $(0, 0)$ in this band. The next theorem shows that we can build the appropriate data structures to maintain the inductive invariant through recursive doubling.

THEOREM 11. *Suppose we have just merged any two nested bands into a new band. Then the front pointers of the new band form a tree and the back pointers of the new band form a linked list.*

There is a proof by induction based on Theorem 9.

Theorem 11 shows that the inductive invariant holds given the appropriate data structures and computations.

4.1. Merging bands using $n^2/\lg n$ processors. In this subsection we show how to merge two bands using $n^2/\lg n$ processors in $O(\lg n)$ time. This algorithm also merges two optimally triangulated convex polygons when all of the weights of one polygon are heavier than all of the weights of the other. Given a triangle with vertices w_i, w_j, and w_k its cost is $w_i w_j w_k$; also see [18, 28].

Recursively doubling the band merging algorithm while using the proper data structures and appropriate tree contracting gives the $n^2/\lg n$-processor and $O(\lg^3 n)$-time MCOP algorithm.

The algorithm in Figure 17 merges two bands in $O(\lg n)$ time using $n^2/\lg n$ processors. Adding the cost of recursive doubling and tree contraction gives a factor of $O(\lg^2 n)$ time to the entire algorithm, making the total cost for solving the MCOP $O(\lg^3 n)$ time using $n^2/\lg n$ processors.

The two **for** loops in step 1 of the algorithm in Figure 17 perform the edge minimizing. This is the only part of this algorithm that uses $n^2/\lg n$ processors. In $O(\lg n)$ time using $n^2/\lg n$ processors we can edge minimize unit paths with contracted trees such as those depicted in the bottleneck of Figure 5c.

The **for** loops in step 2 compute the supercritical nodes of the band that are being created by merging. Step 3 computes the shortest paths forward for all critical nodes in the inner band.

The base case for the recursive doubling can be established by breaking the canonical subgraphs into bands of constant width. Then for each band sequentially, let the $n/\lg n$ processors set up the inductive invariant in $O(\lg n)$ time. Number the nested bands consecutively according to their nestings by the Euler tour technique so the algorithm can track adjacent bands for merging.

The correctness of the algorithm in Figure 18 comes from Theorems 7, 9, and 11.

The time complexity of solving the MCOP can be reduced to $O(\lg^2 n)$ time with $n^2/\lg n$ processors by performing band merging and then tree contraction.

THEOREM 12. *Recursive doubling with band merging can be done simultaneously with tree contraction, thereby solving the MCOP in $O(\lg^2 n)$ time with $n^2/\lg n$ processors.*

Proof. Take any canonical tree T with nontrivial bands and leaves. Then T has at most $n-1$ critical nodes. In general, for any arithmetic expression tree with $n-1$ nodes, it takes $O(\lg n)$ time to contract it. In a canonical tree we have just seen

Take two adjacent nested bands, say $D^{(i,v)}_{(j,t)}$ nested around $D^{(j,t)}_{(k,s)}$, such that for each band individually the inductive invariant holds.

1. **for all** supercritical nodes $(x, y) \in V[\overline{p}^{(j,t)}_{(k,s)}]$ **in parallel do**
   for all angular edges from (x, y) to all $(u, z) \in V[p^{(i,v)}_{(j,t)}]$ **in parallel do**
   find the angular edge between the bands that gives a shortest path from (x, y) all the way to (i, v), compute the *cost-of-front-ptr*s for these new edges
   let each supercritical node (x, y) have a pointer to a shortest path through $p^{(i,v)}_{(j,t)}$ to (i, v)
   for the supercritical nodes in $\overline{p}^{(j,t)}_{(k,s)}$ put the angular edge that gives them a shortest path forward to (i, v) in M
2. **for all** angular edges in M **in parallel do**
   find the shortest path N from (i, v) back to $(0, 0)$ through $D^{(i,v)}_{(k,s)}$
   for all critical nodes in the path N **in parallel do**
   using pointer jumping build the *back-ptr*s giving any new supercritical nodes and compute the values of *cost-to-back* for each new supercritical node
3. **for all** *non*-supercritical nodes in $p^{(j,t)}_{(k,s)}$ **in parallel do**
   using pointer jumping expand the tree of *front-ptr*s through the new angular edges in M and their minimal values to (i, v). This gives trees joined by a linked list through the supercritical nodes.
   With this find the shortest path to (i, v) for all non-supercritical nodes in $p^{(j,t)}_{(k,s)}$ by computing a partial prefix in a rooted tree.
   Also compute all of the new *cost-to-front* values using a parallel partial prefix.

FIG. 17. *An $O(\lg n)$-time and $n^2/\lg n$-processor algorithm for merging two bands.*

that each contraction operation (raking) can be done in $O(\lg n)$ time using $n^2/\lg n$ processors. This is because in the worst case a leaf raking operation in a canonical tree is the merging of two bands. Now, each band can be seen as no more than a linked list in the canonical tree that must be contracted where there is one leaf per list node. Now, we can simply take every band that has k critical nodes and is in any canonical tree, and we can assume that it has 2^c "linked list nodes" such that $2^{c-1} < k \leq 2^c$. With this, each raking operation will cost at most $O(\lg n)$ time using $n^2/\lg n$ processors. In addition, by assuming k is the nearest power of two greater than or equal to k, we are at most doubling the number of critical nodes in T. Hence the asymptotic bound we claim must hold. □

5. An efficient polylog-time MCOP algorithm. In this section we reduce the processor complexity of the band merging algorithm of section 4. The results of this section are based on a parallel divide-and-conquer form of binary search which is tied into some classical problems of finding row minima in totally monotone matrices.

Theorems 8 and 10 supply the basis for a parallel divide-and-conquer binary search algorithm that finds the jumpers that minimize each unit path in a canonical graph.

1. Find the middle supercritical node in the inner band, say $(x-1, y)$.
2. Using $m/\lg m$ processors and in $O(\lg m)$ time find a shortest path forward from $(x-1, y)$ to the front of the outer band. Suppose that this shortest path forward from the supercritical node $(x-1, y)$ has an angular edge between the two bands that terminates in row r.
3. Split the jumpers into two sets:
   (a) Those smaller than or equal to $(r, x) \Longrightarrow (r, y)$; call them S. They are nested *inside* $(r, x) \Longrightarrow (r, y)$.
   (b) Those larger than or equal to $(r, x) \Longrightarrow (r, y)$; call them L. They are nested *around* $(r, x) \Longrightarrow (r, y)$.
4. Do the following two steps in parallel:
   (a) Assign $|S|$ processors to rows r up through 1 and recursively repeat this algorithm with the jumpers in S.
   (b) Assign $|L|$ processors to rows r down through m and recursively repeat this procedure with the jumpers in L.

FIG. 18. *An $O(\lg^2 n)$-time and $n/\lg n$-processor band merging algorithm.*

THEOREM 13. *Suppose that r is the row in the outer band such that the dual of $(r, x) \Longrightarrow (r, y)$ gives a shortest path forward from the supercritical node $(x-1, y)$ of the inner band to the front node of the outer band. Then to find shortest paths forward from other supercritical nodes,*

- *it is sufficient to consider only larger nested jumpers in any row s below row r, that is, $w_s > w_r$, and*
- *it is sufficient to consider only smaller nested jumpers in any row i above row r, that is, $w_i < w_r$.*

A proof of this theorem comes directly from Theorems 8 and 10.

The next algorithm replaces the two nested **for** loops in step 1 in the algorithm of Figure 17. This next algorithm gives shortest paths forward for all supercritical nodes originally in the inner band and a shortest path back to $(0, 0)$ through the two merged bands.

Assuming that each band has m critical nodes, the next procedure finds shortest paths from all supercritical nodes of the inner band to the front of the outer band. In addition, assume the shortest path information before the merging and all shortest paths to the front of the outer band. Then the shortest path back from the front node of the outer band is easily computed. As before, begin assuming the inductive invariant. Also, all jumpers in the next algorithm are jumpers that get their *sp* values from the inner band where the jumpers themselves are in unit rows or columns of the outer band.

The following algorithm is strikingly similar to those discussed in [1] and [2]. This key observation leads to some complexity improvements.

Now assign one processor to each unit path in the outer band. For each unit path, summing the cost to the critical node and the cost from the critical node to the front supercritical node of the outer band gives a shortest path backwards from the front node of the outer band to $(0, 0)$. These minimal paths can be computed in $O(\lg n)$ time using $n/\lg n$ processors. If a unit path has no edge minimizing jumpers, then this algorithm just finds the shortest path forward for all supercritical nodes in the

inner band, since, in this case, the shortest path back to $(0, 0)$ from the front critical node of the outer band does not go through the inner band.

The algorithm in Figure 18 also breaks through the bottleneck of Figure 5c. It takes $O(\lg^2 n)$ time and uses $n/\lg n$ processors in the worst case. Considering the cost of the recursive doubling and the tree contraction gives the $O(\lg^3 n)$-time and $n/\lg n$-processor matrix chain ordering algorithm.

The next corollary shows that the algorithm given here can be improved by using efficient algorithms for finding row minima in totally monotone matrices. A $m \times n$ matrix M is *totally monotone* if every 2×2 submatrix is monotone. That is, for all $1 \leq i < k \leq m$ and $1 \leq j < l \leq n$, if $M[i, j] > M[i, l]$, then $M[k, j] > M[k, l]$.

This row minima problem is classical and has been shown to be at the root of many important problems; see for example [1, 2].

COROLLARY 3. *Solving the row minima problem on totally monotone matrices allows us to merge two bands.*

Proof. Given two nested bands to merge, for ease of exposition take only the horizontal straight unit paths of the outer band. Let each of these straight unit paths denote the row of a matrix M. Each column of M represents the jumpers that get their *sp* values from the supercritical nodes of the inner band. The first column represents the effect of the innermost jumper, the second column represents the effect of the immediate jumper containing it, etc. Similarly, several sets of independent jumpers give several totally monotone matrices.

By Theorem 13, M is a monotone matrix. But, any submatrix of M represents neighboring straight unit paths in the rows and neighboring jumpers along the columns. Similarly, every 2×2 submatrix is monotone. Since Theorems 8 and 10 still hold, we know that such a submatrix is also monotone since we can again apply Theorem 13. □

Therefore, our algorithm is one of the many known to depend on the row minima problem on a totally monotone matrix. Hence, by the results of Aggarwal and Park [2] and Atallah and Kosaraju [6], our algorithm runs in $O(\lg^2 n \lg\lg n)$ time using $n/\lg\lg n$ processors on a common CRCW PRAM, or in $O(\lg^2 n)$ time using n processors on an EREW PRAM. For the EREW PRAM algorithm note that from pointer jumping to tree contraction the time complexity stays the same asymptotically.

An asymptotically optimal polylog-time row minima algorithm for totally monotone matrices would make the work of our MCOP algorithm the same as the work of Hu and Shing's $O(n \lg n)$ sequential algorithm. Very recently Bradford, Fleischer, and Smid [14] give an algorithm for computing the row minima of totally monotone matrices with $O(n\sqrt{\lg n})$ work and $O(\lg n \lg\lg n)$ time on a CREW PRAM (and several variations on other PRAM models). The results of [14] lead to an $O(n \lg^{1.5} n)$ work and polylog time CREW PRAM algorithm for the MCOP.

Hu and Shing's algorithm has the best known work for solving the MCOP to date [28, 29]. In this regard, in [38, 39] Ramanan shows that problems closely related to the MCOP have a $\Omega(n \lg n)$ lower bound. Furthermore, in [13] Bradford, Choppella, and Rawlins give several lower bounds for the MCOP on different models of computation, including a simple $\Omega(n \lg n)$ lower bound on the comparison based model for a constrained version of the MCOP.

6. Conclusions. The study of efficient parallel algorithms for problems with elementary dynamic programming solutions is rich with interesting results. This paper gives an algorithm that solves the matrix chain ordering problem to within less than

a log factor of the best serial solution. Furthermore, the best serial solution is in some sense optimal. This algorithm also solves a problem of finding an optimal triangulation of a convex polygon.

Acknowledgments. We acknowledge conversations with Alok Aggarwal, Venkatesh Choppella, Artur Czumaj, Ming Kao, Larry Larmore, and Kunsoo Park that were very helpful. In addition, conversations with Danny Chen were highlighted when, at Midwest Theory Day, he pointed out reference [16] which helps make our algorithm more efficient on the EREW PRAM.

REFERENCES

[1] A. Aggarwal, M. M. Klawe, S. Moran, P. Shor, and R. Wilbur, *Geometric applications of a matrix searching algorithm*, Algorithmica, 2 (1987), pp. 195–208.

[2] A. Aggarwal and J. Park, *Parallel searching multidimensional monotone arrays*, J. Algorithms, to appear; in Proc. 29th Annual IEEE Symp. on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 497–512.

[3] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison–Wesley, Reading, MA, 1974.

[4] R. Anderson and E. W. Mayr, *Parallelism and the maximal path problem*, Inform. Process. Lett., 24 (1978), pp. 121–126.

[5] A. Apostolico, M. J. Atallah, L. L. Larmore, and S. H. McFaddin, *Efficient parallel algorithms for string editing and related problems*, SIAM J. Comput., 19 (1990), pp. 968–988.

[6] M. J. Atallah and S. R. Kosaraju, *An efficient parallel algorithm for the row minima of a totally monotone matrix*, J. Algorithms, 13 (1992), pp. 394–413.

[7] M. J. Atallah, S. R. Kosaraju, L. L. Larmore, G. L. Miller, and S.-H. Teng, *Constructing trees in parallel*, in Proc. 1st ACM Symp. on Parallel Algorithms and Architectures, ACM, New York, 1989, pp. 499–533.

[8] O. Berkman, D. Breslauer, Z. Galil, B. Schieber, and U. Vishkin, *Highly parallelizable problems*, in Proc. 21st Annual ACM Symposium on the Theory of Computing, ACM, New York, 1989, pp. 309–319.

[9] O. Berkman, B. Schieber, and U. Vishkin, *Optimal doubly logarithmic parallel algorithms based on finding all nearest smaller values*, J. Algorithms, 14 (1993), pp. 344–370.

[10] P. G. Bradford, *Efficient Parallel Dynamic Programming*, Technical Report 352, Indiana University, Bloomington, IN, April 1992 and October 1994 (revised).

[11] P. G. Bradford, *Efficient Parallel Dynamic Programming*, extended abstract in the Proceedings of the 30th Allerton Conference on Communication, Control and Computation, University of Illinois at Urbana-Champaign, 1992, pp. 185–194. Full version: Technical Report 352, Indiana University, Bloomington, IN, April 1992 and October 1994 (revised).

[12] P. G. Bradford, *Efficient Parallel Dynamic Programming*, Ph.D. dissertation, February 1995, Indiana University, Bloomington, IN; also Parallel Dynamic Programming Technical Report TR 424, February 1995, Indiana University.

[13] P. G. Bradford, V. Choppella, and G. J. E. Rawlins, *Lower bounds for the matrix chain ordering problem (extended abstract)*, in the Proceedings of LATIN '95: Theoretical Informatics, Lecture Notes in Comp. Sci. 911, R. Baeza-Yates, E. Goles, and P. V. Poblete, eds., Springer-Verlag, New York, 1995, pp. 112–130.

[14] P. G. Bradford, R. Fleischer, and M. Smid, *More efficient parallel totally monotone matrix searching*, J. Algorithms, 23 (1997), pp. 386–400.

[15] P. G. Bradford, G. J. E. Rawlins, and G. E. Shannon, *Matrix Chain Ordering in Polylog Time with $n/\lg n$ Processors*, Technical Report 360, Indiana University, Bloomington, IN, December 1992.

[16] D. Z. Chen, *Efficient geometric algorithms on the EREW PRAM*, in Proceedings of the 28th Allerton Conference on Communication, Control, and Computation, Monticello, IL, 1990, pp. 818–827. Full version in IEEE Trans. Parallel Distrib. Systems, 6 (1995), pp. 41–47.

[17] F. Y. Chin, *An $O(n)$ algorithm for determining near-optimal computation order of matrix chain products*, Comm. ACM, 21 (1978), pp. 544–549.

[18] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, McGraw–Hill, New York, 1990.
[19] A. Czumaj, *An optimal parallel algorithm for computing a near-optimal order of matrix multiplications*, in SWAT 92, Lecture Notes in Comput. Sci. 621, Springer-Verlag, New York, 1992, pp. 62–72.
[20] A. Czumaj, *Parallel algorithm for the matrix chain product and the optimal triangulation problem*, in STACS 93, Lecture Notes in Comput. Sci. 665, Springer-Verlag, New York, 1993, pp. 294–305.
[21] L. E. Deimel, Jr. and T. A. Lampe, *An Invariance Theorem Concerning Optimal Computation of Matrix Chain Products*, Technical Report TR79-14, North Carolina State Univ., Raleigh, NC.
[22] Z. Galil and K. Park, *Parallel algorithms for dynamic programming recurrences with more than $O(1)$ dependency*, J. Parallel Distrib. Comput., 21 (1994), pp. 213–222.
[23] A. Gibbons and W. Rytter, *Efficient Parallel Algorithms*, Cambridge University Press, Cambridge, 1988.
[24] D. Hillis and G. L. Steele, Jr., *Data parallel algorithms*, Comm. ACM, 29 (1986), pp. 1170–1183.
[25] S.-H. S. Huang, H. Liu, and V. Viswanathan, *Parallel dynamic programming*, in Proc. 2nd Annual IEEE Symposium on Parallel and Distributed Processing, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 497–500.
[26] S.-H. S. Huang, H. Liu, and V. Viswanathan, *A sublinear parallel algorithm for some dynamic programming problems*, Theoret. Comput. Sci., 106 (1992), pp. 361–371.
[27] T. C. Hu and M. T. Shing, *An $O(n)$ algorithm to find a near-optimum partition of a convex polygon*, J. Algorithms, 2 (1981), pp. 122–138.
[28] T. C. Hu and M. T. Shing, *Computation of matrix product chains. Part* I, SIAM J. Comput., 11 (1982), pp. 362–373.
[29] T. C. Hu and M. T. Shing, *Computation of matrix product chains. Part* II, SIAM J. Comput., 13 (1984), pp. 228–251.
[30] O. H. Ibarra, T.-C. Pong, and S. M. Sohn, *Hypercube algorithms for some string comparison problems*, in Proc. IEEE International Conference on Parallel Processing, IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 190–193.
[31] J. JáJá, *An Introduction to Parallel Algorithms*, Addison–Wesley, Reading, MA, 1992.
[32] D. G. Kirkpatrick and T. Przytycka, *Parallel construction of near optimal binary search trees*, in Proc. 2nd ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, 1990, pp. 234–243.
[33] R. M. Karp and V. Ramachandran, *Parallel algorithms for shared-memory machines*, in Handbook of Theoretical Computer Science, Vol. A, V. Van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 869–941.
[34] S. K. Kim, *Optimal Parallel Algorithms on Sorted Intervals*, Technical Report TR 90-01-04, Department of Computer Science and Engineering, University of Washington, Seattle, WA, 1990.
[35] P. N. Klein and J. H. Reif, *Parallel time $O(\lg n)$ acceptance of deterministic CFLs on an exclusive-write P-RAM*, SIAM J. Comput., 17 (1988), pp. 463–485.
[36] V. Kumar, A. Grama, A. Gupta, and G. Kraypis, *Introduction to Parallel Computing*, Benjamin/Cummings, Redwood City, CA, 1994.
[37] L. L. Larmore and W. Rytter, *Efficient sublinear time parallel algorithms for the recognition of context-free languages*, in SWAT 91, Lecture Notes in Comput. Sci. 577, Springer Verlag, New York, 1991, pp. 121–132.
[38] P. Ramanan, *A new lower bound technique and its application: Tight lower bounds for a polygon triangularization problem*, in Proc. 2nd Annual ACM–SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1991, pp. 281–290.
[39] P. Ramanan, *A new lower bound technique and its application: Tight lower bounds for a polygon triangularization problem*, SIAM J. Comput., 23 (1994), pp. 834–851.
[40] P. Ramanan, *An Efficient Parallel Algorithm for Finding an Optimal Order of Computing a Matrix Chain Product*, Technical Report WSUCS-92-2, Wichita State University, Wichita, KS, June 1992.
[41] P. Ramanan, *An Efficient Parallel Algorithm for the Matrix Chain Product Problem*, Technical Report WSUCS-93-1, Wichita State University, Wichita, KS, January 1993.
[42] W. Rytter, *On efficient parallel computation for some dynamic programming problems*, Theoret. Comput. Sci., 59 (1988), pp. 297–307.
[43] L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff, *Fast parallel computation of polynomials using few processors*, SIAM J. Comput., 12 (1983), pp. 641–644.
[44] F. F. Yao, *Speed-up in dynamic programming*, SIAM J. Algebraic Discrete Methods, 3 (1982), pp. 532–540.

SIAM J. COMPUT.
Vol. 27, No. 2, pp. 491–505, April 1998
© 1998 Society for Industrial and Applied Mathematics
009

COMPUTING MANY FACES IN ARRANGEMENTS OF LINES AND SEGMENTS*

PANKAJ K. AGARWAL†, JIŘÍ MATOUŠEK‡, AND OTFRIED SCHWARZKOPF§

Abstract. We present randomized algorithms for computing many faces in an arrangement of lines or of segments in the plane, which are considerably simpler and slightly faster than the previously known ones. The main new idea is a simple randomized $O(n \log n)$ expected time algorithm for computing $\sqrt{n}$ cells in an arrangement of n lines.

Key words. arrangements, random sampling, duality

AMS subject classifications. 68Q20, 68R05, 68U05

PII. S009753979426616X

1. Introduction. Given a finite set S of lines in the plane, the *arrangement* of S, denoted by $\mathcal{A}(S)$, is the cell complex induced by S. The 0-faces (or *vertices*) of $\mathcal{A}(S)$ are the intersection points of S, the 1-faces (or *edges*) are maximal portions of lines of S that do not contain any vertex, and the 2-faces (called *cells*) are the connected components of $\mathbb{R}^2 - \bigcup S$. For a finite set S of segments we define the arrangement, $\mathcal{A}(S)$, in an analogous manner. Notice that while the cells are convex in an arrangement of lines, they need not even be simply connected in an arrangement of segments.

Line and segment arrangements have been extensively studied in computational geometry (as well as in some other areas), as a wide range of computational geometry problems can be formulated in terms of computing such arrangements or their parts [11, 14].

Given a set S of n lines and a set P of m points in the plane, we define $\mathcal{A}(S, P)$ to be the collection of all cells of $\mathcal{A}(S)$ that contain at least one point of P. The *combinatorial complexity* of a cell C in $\mathcal{A}(S)$, denoted by $|C|$, is the number of edges of C. Let $\kappa(S, P) = \sum_{C \in \mathcal{A}(S,P)} |C|$ denote the total combinatorial complexity of all cells in $\mathcal{A}(S, P)$, and let

$$\kappa(n, m) = \max \kappa(S, P)\,,$$

where the maximum is taken over all sets of n lines and over all sets of m points in

*Received by the editors April 18, 1994; accepted for publication (in revised form) March 11, 1996. A part of this work was done while the first and third authors were visiting Charles University and while the first author was visiting Utrecht University. The first author has been supported by National Science Foundation grant CCR–93–01259 and an NYI award. The second author has been supported by Charles University grant 351 and Czech Republic grant GAČR 201/93/2167. The third author has been supported by the Netherlands' Organization for Scientific Research (NWO) and partially supported by ESPRIT Basic Research Action 7141 (project ALCOM II: *Algorithms and Complexity*).

http://www.siam.org/journals/sicomp/27-2/26616.html

†Department of Computer Science, Box 90129, Duke University, Durham, NC 27708-0129 (pankaj@euclid.cs.duke.edu).

‡Department of Applied Mathematics, Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic (matousek@kam.mff.cuni.cz).

§Department of Computer Science, Utrecht University, P. O. Box 80.089, 3508 TB Utrecht, the Netherlands. Current address: Department of Computer Science, Hong Kong University of Science & Technology, Clear Water Bay, Kowloon, Hong Kong.

the plane. It is known that

$$\kappa(n,m) = \Theta(n^{2/3}m^{2/3} + n + m)\,.$$

The upper bound was proven by Clarkson et al. [9] and the lower bound follows from a result of Szemerédi and Trotter [21]; previous results and related work can be found in Canham [4] and Edelsbrunner and Welzl [13].

In this paper we study the problem of computing $\mathcal{A}(S,P)$, that is, for each cell $C \in \mathcal{A}(S,P)$, we want to return the vertices of C in, say, clockwise order. We will refer to the cells of $\mathcal{A}(S,P)$ as the *marked* cells of $\mathcal{A}(S)$. Edelsbrunner, Guibas, and Sharir [12] presented a randomized algorithm, based on the random-sampling technique [16], for computing $\mathcal{A}(S,P)$, whose expected running time was

$$O(m^{2/3-\varepsilon}n^{2/3+2\varepsilon}\log n + n\log n\log m),$$

for any fixed $\varepsilon > 0$. A deterministic algorithm with running time

$$O(m^{2/3}n^{2/3}\log^{O(1)} n + n\log^3 n + m\log n)$$

was given by Agarwal [1]. These algorithms thus are nearly worst-case optimal, but both of them are rather involved.

Recently, randomized incremental algorithms have been developed for a variety of geometric problems, which add the input objects one by one in random order and maintain the desired structure; see e.g., [6, 10, 18, 19]. In our case, we can add the lines of S one by one in random order and maintain the marked cells in the arrangement of lines added so far. However, the expected running time of this approach is $\Omega(n\sqrt{m} + m\log n)$ in the worst case. We, therefore, do not quite follow the randomized incremental paradigm.

We begin by presenting an expected $O((m^2+n)\log n)$-time randomized algorithm for computing $\mathcal{A}(S,P)$. Notice that for $m \le \sqrt{n}$, the expected running time of the algorithm is $O(n\log n)$, which matches the known lower bound for computing a single cell in line arrangements. We then apply the randomized geometric divide-and-conquer technique in a standard way, obtaining an algorithm with expected time

$$O\left(m^{2/3}n^{2/3}\log\frac{n}{\sqrt{m}} + (m+n)\log n\right).$$

We also study a similar but more difficult problem of computing the marked cells in an arrangement of n line segments. Let S be a set of n segments in the plane. We use an analogous notation $\mathcal{A}(S,P)$ to denote the set of cells in $\mathcal{A}(S)$ containing at least one point of P, and $\eta(n,m)$ to denote the maximum combinatorial complexity of $\mathcal{A}(S,P)$ over all sets S of n segments and over all sets P of m points in the plane. Aronov et al. [2] proved that

$$\eta(n,m) = O\left(m^{2/3}n^{2/3} + n\log m + n\,\alpha(n)\right).$$

A randomized algorithm with expected running time

$$O(m^{2/3-\varepsilon}n^{2/3+2\varepsilon}\log n + n\alpha(n)\log^2 n\log m)$$

is described by Edelsbrunner, Guibas, and Sharir [12], and a slightly faster deterministic algorithm is presented by Agarwal [1]. See [20] for results on computing a single cell in arrangements of segments.

Following the same strategy as for the case of lines, we first develop a randomized algorithm with $O((m^2 \log m + n \log m + n\,\alpha(n)) \log n)$ expected running time. Note that the above upper bound for $\eta(n, m)$ is not known to be tight, and a bound such as $\eta(n, \sqrt{n}) = O(n\,\alpha(n))$ (which is conjectured to be the complexity of $\sqrt{n}$ cells) will immediately improve the expected running time of our algorithm to $O(n \log n\,\alpha(n))$. Plugging this algorithm into the standard random-sampling technique, as in the case of lines, we obtain a randomized algorithm for computing $\mathcal{A}(S, P)$ whose expected running time is

$$O\left(m^{2/3}n^{2/3} \log^2 \frac{n}{\sqrt{m}} + (m + n \log m + n\,\alpha(n)) \log n\right).$$

If the segments of S have only $k = o(n^2)$ intersection points, the expected running time of the algorithm is

$$O\left(m^{2/3}k^{1/3} \log^2 \frac{k}{m} + (m + n \log m + n\,\alpha(n)) \log n\right).$$

For the analysis of the expected running time of our algorithms we will use a generalization of a lemma due to Chazelle and Friedman [7]. (An alternative analysis could probably be obtained using a method similar to that of Chazelle et al. [6], but we hope that our approach is somewhat more intuitive).

2. A generalization of the Chazelle–Friedman lemma. Let S be a set of lines or segments and P a set of points in the plane. For a cell C in the collection $\mathcal{A}(S, P)$, let $C^{\|}$ denote the collection of trapezoids in the vertical decomposition[1] of C, and let $\mathcal{A}^{\|}(S, P) = \bigcup_{C \in \mathcal{A}(S,P)} C^{\|}$ denote the set of trapezoids in the vertical decomposition of $\mathcal{A}(S, P)$. Abusing the notation slightly, we will use $\mathcal{A}^{\|}(S, P)$ to denote the corresponding planar subdivision as well. For any subset $X \subseteq S$ and any trapezoid $\Delta \in \mathcal{A}^{\|}(X, P)$, let $w(\Delta)$ denote the number of elements of S intersecting the interior of Δ.

Let $n = |S|$, and let R be a random subset of S of size r. For the analysis of our algorithms, we are interested in estimating the expectation, over all random choices of R,

$$\mathrm{E}\left[\sum_{\Delta \in \mathcal{A}^{\|}(R,P)} w(\Delta)^c\right], \tag{2.1}$$

where c is a small constant; for instance, $c = 2$. Well-known results concerning the so-called ε-nets (Haussler and Welzl [16]) imply that, for every $\Delta \in \mathcal{A}^{\|}(R)$, $w(\Delta) \leq a(n/r) \log r$ with high probability, where a is a suitable constant. This inequality can be used to derive a weaker bound for (2.1). We are, however, interested in the following, slightly stronger bound (better by a factor of $\log^c r$).

PROPOSITION 2.1. (i) *Let S be a set of n lines and P a set of m points in the plane. If $R \subseteq S$ is a random subset of size r, where each subset of size r is chosen with equal probability, then for any constant $c \geq 1$,*

$$\mathrm{E}\left[\sum_{\Delta \in \mathcal{A}^{\|}(R,P)} w(\Delta)^c\right] = \kappa(r, m) \cdot O((n/r)^c).$$

[1]The vertical decomposition $C^{\|}$ of a cell C in an arrangement of segments (or of lines) is obtained by drawing a vertical line segment from each vertex of C in both directions (within C) until it hits another edge of C. If the segment does not intersect any edge of C, then it is extended to infinity.

(ii) *Let S be a set of n segments and P a set of m points in the plane. If $R \subseteq S$ is a random subset of size r, where each subset of size r is chosen with equal probability, then for any constant $c \geq 1$,*

$$\mathrm{E}\left[\sum_{\Delta \in \mathcal{A}^{\parallel}(R,P)} w(\Delta)^c\right] = \eta(r,m) \cdot O((n/r)^c)\,.$$

These bounds essentially say that the cth moment of the quantities $w(\Delta)$ behaves as if $w(\Delta)$ were $O(n/r)$. If we sum $w(\Delta)$ over all cells in $\mathcal{A}(R)$—the case where every cell of $\mathcal{A}(R)$ contains a point of P—then Proposition 2.1 follows from a result of Clarkson and Shor [10]. In our situation, where the sum is taken over only some of the cells, the Clarkson–Shor framework does not apply directly anymore (the main distinction between these two situations will be outlined below). We give a proof based on a generalization of the approach by Chazelle and Friedman [7], which is somewhat different from the Clarkson–Shor method. Recently, de Berg, Dobrindt, and Schwarzkopf [3] gave an alternative proof of Proposition 2.1.

We derive a key lemma in a somewhat abstract framework; see also [6, 7, 10] for various approaches to axiomatize similar situations.

Let S be a set of objects. For a subset $R \subseteq S$, we define a collection of "regions" called $\mathrm{CT}(R)$; in Proposition 2.1 the objects are lines or segments, the regions are trapezoids, and $\mathrm{CT}(R) = \mathcal{A}^{\parallel}(R,P)$. Let $T = T(S) = \bigcup_{R \subseteq S} CT(R)$ denote the set of regions defined by all possible subsets of S. We associate two subsets $D(\Delta), K(\Delta) \subseteq S$ with each region $\Delta \in T$.

$D(\Delta)$, called the *defining set*, is a subset of S defining the region Δ in a suitable geometric sense.[2] We assume that for every $\Delta \in T$, $|D(\Delta)| \leq d$ for a (small) constant d. In Proposition 2.1, each trapezoid Δ is defined by at most four segments (or lines) of S, which constitute the set $D(\Delta)$; details can be found in Chazelle et al. [6].

$K(\Delta)$, called the *killing set*, is a set of objects of S such that including any object of $K(\Delta)$ into R prevents Δ from appearing in $\mathrm{CT}(R)$. In many applications, $K(\Delta)$ is the set of objects intersecting the cell Δ; this is also the case in Proposition 2.1. Set $w(\Delta) = |K(\Delta)|$.

Let $S, \mathrm{CT}(R), D(\Delta)$, and $K(\Delta)$ be such that for any subset $R \subseteq S$, $\mathrm{CT}(R)$ satisfies the following axioms:

(i) For any $\Delta \in \mathrm{CT}(R)$, $D(\Delta) \subseteq R$ and $R \cap K(\Delta) = \emptyset$, and
(ii) If $\Delta \in \mathrm{CT}(R)$ and R' is a subset of R with $D(\Delta) \subseteq R'$, then $\Delta \in \mathrm{CT}(R')$.

It is easily checked that these axioms hold in the situations of Proposition 2.1.

For any natural number t, let

$$\mathrm{CT}_t(R) = \{\Delta \in \mathrm{CT}(R) \mid w(\Delta) \geq tn/r\}\,.$$

We establish the following.

LEMMA 2.2. *Given a set S of n objects, let R be a random sample of size $r \leq n$ drawn from S, and let t be a parameter, $1 \leq t \leq r/d$, where $d = \max|D(\Delta)|$. Assuming that $\mathrm{CT}(R)$, $D(\Delta)$, and $K(\Delta)$ satisfy axioms* (i) *and* (ii) *above, we have*

$$\mathrm{E}\left[|\mathrm{CT}_t(R)|\right] = O(2^{-t}) \cdot \mathrm{E}\left[|\mathrm{CT}(R')|\right], \tag{2.2}$$

where $R' \subseteq S$ denotes a random sample of size $r' = \lfloor r/t \rfloor$.

[2] We need not make this precise here, as this is only an intuitive meaning of $D(\Delta)$. The analysis depends only on the axioms involving $D(\Delta)$ given below, and these will be satisfied in our specific applications.

Roughly speaking, Lemma 2.2 says that the expected number of "large" trapezoids in $\mathrm{CT}(R)$, that is, trapezoids for which the value of $w(\Delta)$ exceeds the "right" value n/r more than t times, decreases exponentially with t.

Chazelle and Friedman [7] proved a result analogous to Lemma 2.2 under the following stronger axiom replacing (ii):

(ii′) If $D(\Delta) \subseteq R$ and $K(\Delta) \cap R = \emptyset$, then $\Delta \in \mathrm{CT}(R)$.

This assumption implies that the presence of Δ in $\mathrm{CT}(R)$ depends only on $D(\Delta)$ and $K(\Delta)$; thus it is determined purely "locally." Notice that (ii′) may fail in the situation of Proposition 2.1. However, (ii′) holds in the special case, when $\mathrm{CT}(R)$ is the vertical decomposition of *all* cells in $\mathcal{A}(R)$.

Proof of Lemma 2.2. Let $T_t = \bigcup_{R \subseteq S} \mathrm{CT}_t(R)$. We have

$$\mathrm{E}\left[|\mathrm{CT}_t(R)|\right] = \sum_{\Delta \in T_t} \Pr[\Delta \in \mathrm{CT}(R)]\,, \tag{2.3}$$

$$\begin{aligned}\mathrm{E}\left[|\mathrm{CT}(R')|\right] &= \sum_{\Delta \in T} \Pr[\Delta \in \mathrm{CT}(R')] \\ &\geq \sum_{\Delta \in T_t} \Pr[\Delta \in \mathrm{CT}(R')]\,. \end{aligned} \tag{2.4}$$

We will prove that, for each $\Delta \in T_t$,

$$\Pr[\Delta \in \mathrm{CT}(R)] = O(2^{-t}) \cdot \Pr[\Delta \in \mathrm{CT}(R')]\,, \tag{2.5}$$

which in conjunction with (2.3) and (2.4) implies (2.2).

Let A_Δ denote the event $D(\Delta) \subseteq R$ and $K(\Delta) \cap R = \emptyset$, and let A'_Δ denote the event $D(\Delta) \subseteq R'$ and $K(\Delta) \cap R' = \emptyset$.

We rewrite $\Pr[\Delta \in \mathrm{CT}(R)]$ using the following definition of conditional probability:

$$\Pr[\Delta \in \mathrm{CT}(R)] = \Pr[A_\Delta] \cdot \Pr[\Delta \in \mathrm{CT}(R) \mid A_\Delta]$$

and analogously,

$$\Pr[\Delta \in \mathrm{CT}(R')] = \Pr[A'_\Delta] \cdot \Pr[\Delta \in \mathrm{CT}(R') \mid A'_\Delta]\,.$$

We observe that, by axiom (ii), we have

$$\Pr[\Delta \in \mathrm{CT}(R) \mid A_\Delta] \leq \Pr[\Delta \in \mathrm{CT}(R') \mid A'_\Delta]\,. \tag{2.6}$$

To see this, consider the following random experiment. Select a set R' by including all the elements of $D(\Delta)$ into it, and adding $r' - |D(\Delta)|$ elements chosen randomly among the elements of $S \setminus (D(\Delta) \cup K(\Delta))$. By definition of the conditional probability, the probability that Δ appears in $\mathrm{CT}(R')$ is precisely $\Pr[\Delta \in \mathrm{CT}(R') \mid A'_\Delta]$. Now take this R' and add $r - r'$ randomly chosen elements of $S \setminus (R' \cup K(\Delta))$ to it, obtaining a set R. The distribution of this R is clearly the same as if we took the elements of $D(\Delta)$ and added $r - |D(\Delta)|$ random elements of $S \setminus (D(\Delta) \cup K(\Delta))$, and hence the probability of $\Delta \in \mathrm{CT}(R)$ is $\Pr[\Delta \in \mathrm{CT}(R) \mid A_\Delta]$. On the other hand, since R was created by adding extra elements to R', whenever Δ is present in $\mathrm{CT}(R)$ it must be in $\mathrm{CT}(R')$ as well, and thus (2.6) holds.

Therefore,

$$\frac{\Pr[\Delta \in \mathrm{CT}(R)]}{\Pr[\Delta \in \mathrm{CT}(R')]} \le \frac{\Pr[A_\Delta]}{\Pr[A'_\Delta]}.$$

(Note that $r' = \lfloor r/t \rfloor \ge d$, and hence both denominators are nonzero.)

It remains to estimate the latter ratio, which can be done in the same way as by Chazelle and Friedman. Let $\delta = |D(\Delta)| \le d$, $w = w(\Delta)$, and for two nonnegative integers $a \le x$, let $x^{\underline{a}} = x(x-1)\cdots(x-a+1)$. Then

$$\begin{aligned}\frac{\Pr[A_\Delta]}{\Pr[A'_\Delta]} &= \frac{\binom{n-w-\delta}{r-\delta}}{\binom{n}{r}} \cdot \frac{\binom{n}{r'}}{\binom{n-w-\delta}{r'-\delta}} \\ &\le \frac{r^{\underline{d}}}{r'^{\underline{d}}} \cdot \frac{(n-w-r')^{\underline{r-r'}}}{(n-r')^{\underline{r-r'}}}.\end{aligned}$$

By our assumption, $r' \ge d$, so we obtain

$$\frac{r-i}{r'-i} \le dt \quad \text{for } i = 0, 1, \ldots, d-1.$$

Thus, the first factor in the above expression is $O(t^d)$. To bound the second factor, we observe that, for $i = r', r'+1, \ldots, r-1$,

$$\frac{n-w-i}{n-i} = 1 - \frac{w}{n-i} \le 1 - \frac{w}{n} \le \exp(-w/n).$$

Since $w \ge tn/r$, we have $w/n \ge t/r$, and therefore,

$$\begin{aligned}\frac{\Pr[A_\Delta]}{\Pr[A'_\Delta]} &\le O(t^d)\exp\left(\frac{-t(r-r')}{r}\right) \\ &= O(t^d)\exp(-(t-1)) = O(2^{-t}),\end{aligned}$$

as desired. □

We now prove Proposition 2.1.

Proof of Proposition 2.1. We will only prove the first part; the second part is identical. For any subset $R \subseteq S$ of size r, let $CT(R)$ denote the set of trapezoids in the vertical decomposition of the marked cells of $\mathcal{A}(R)$, i.e., $CT(R) = \mathcal{A}^{\shortparallel}(R, P)$. Obviously, $|CT(R)| \le \kappa(r, m)$. Now

$$\begin{aligned}&\mathrm{E}\left[\sum_{\Delta \in \mathcal{A}^{\shortparallel}(R,P)} w(\Delta)^c\right] \\ &= \mathrm{E}\left[\sum_{t \ge 1} \left(t\frac{n}{r}\right)^c \left(|CT_t(R)| - |CT_{t-1}(R)|\right)\right] \\ &\le \left(\frac{n}{r}\right)^c \sum_{t \ge 0} (t+1)^c \cdot \mathrm{E}\left[|CT_t(R)|\right] \\ &= \left(\frac{n}{r}\right)^c \sum_{t \ge 0} t^c O(2^{-t}) \cdot \kappa\left(\frac{r}{t}, m\right) \\ &\le \kappa(r,m)\left(\frac{n}{r}\right)^c \sum_{t \ge 0} O(t^c \cdot 2^{-t}) \\ &= \kappa(r,m) \cdot O\left((n/r)^c\right). \qquad \square\end{aligned}$$

3. Computing cells in line arrangements. Let S be a set of n lines and P a set of m points in the plane. We assume that the points of P are sorted in nondecreasing order of their x-coordinates, and that the lines of S are sorted by their slopes. In this section we describe a randomized algorithm for computing $\mathcal{A}(S,P)$. In fact, it computes the vertical decomposition $\mathcal{A}^{\|}(S,P)$ of $\mathcal{A}(S,P)$. Each face of $\mathcal{A}^{\|}(S,P)$ is a trapezoid, bounded by at most two vertical segments and portions of at most two edges of a cell of $\mathcal{A}^{\|}(S,P)$. We begin by presenting a very simple randomized algorithm for computing $\mathcal{A}^{\|}(S,P)$ with $O((m^2+n)\log n)$ expected time, which we will use as a subroutine in the main algorithm. This algorithm is optimal for $m \le \sqrt{n}$. If $n \le n_0$, where n_0 is an appropriate constant, the algorithm computes the vertical decomposition of the entire arrangement using any standard algorithm. Otherwise, it proceeds as follows.

1. Let t be a sufficiently large constant. Choose a random subset $R \subseteq S$ of $r = \lfloor n/t \rfloor$ lines.
2. Partition P into $q = \lceil \sqrt{t}\, \rceil$ subsets $P_1, \ldots, P_q$, each of size at most $k = \lfloor m/\sqrt{t} \rfloor$, where
$$P_i = \{p_{(i-1)k+1}, \ldots, p_{ik}\} \quad \text{for } i < q,$$
$$P_q = \{p_{(q-1)k+1}, \ldots, p_m\}.$$
3. For each $i \le q$, compute $\mathcal{A}^{\|}(R, P_i)$ recursively. If a cell C of $\mathcal{A}(R)$ is computed more than once, retain only one copy of C. (Note that multiple copies of a cell C are computed if C contains the points of more than one P_i.) Since P is sorted in the x-direction, it is easy to detect multiple copies of a cell. In this way, we obtain $\mathcal{A}^{\|}(R,P)$.
4. For each line $\ell \in S \setminus R$, compute the cells of $\mathcal{A}(R,P)$ that ℓ intersects.
5. For each trapezoid Δ of $\mathcal{A}^{\|}(R,P)$, compute the set $S_\Delta \subseteq S \setminus R$ of lines that intersect the interior of Δ.
6. For each trapezoid $\Delta \in \mathcal{A}^{\|}(R,P)$, compute the arrangement of lines of S_Δ, clip it within Δ, and compute the vertical decomposition of the clipped arrangement. For each cell $C \in \mathcal{A}(R,P)$, perform a graph search on trapezoids of these vertical decompositions to merge appropriate trapezoids and to discard superfluous ones, thus forming the portion of $\mathcal{A}^{\|}(S,P)$ within the cell C.

Steps 1–3 are trivial, so we only describe steps 4–6 in more detail.

Step 4. We want to compute the cells of $\mathcal{A}(R,P)$ intersected by each line in $S \setminus R$. The situation can be viewed as follows: we have a collection $\mathcal{C}$ of disjoint convex polygons (the cells of $\mathcal{A}(R,P)$), and a set $S \setminus R$ of lines. The collection $\mathcal{C}$ has at most m polygons with a total of $O(n+m^2)$ edges.[3] For each polygon $C \in \mathcal{C}$, consider C^*, the set of points that are dual to the lines intersecting C. C^* is a polygonal region, bounded by an infinite convex chain from above and by an infinite concave chain from below. Each vertex of C^* is dual to the line supporting an edge of C. For a pair of polygons $C_1, C_2 \in \mathcal{C}$, an intersection point of the edges of C_1^*, C_2^* is dual to a common tangent of C_1 and C_2. Since C_1, C_2 are disjoint, the boundaries of C_1^*, C_2^* intersect in at most four points.

Consider the arrangement $\mathcal{A}(\mathcal{C}^*)$ of the polygonal chains bounding the regions C^*, for all $C \in \mathcal{C}$. It has $O(n+m^2)$ complexity and can be computed in expected time

[3]The latter estimate follows from the bound for $\kappa(n,m)$ mentioned in Section 1, in fact it is the weaker bound proved by Canham [4].

$O((m^2+n)\log n)$, using a randomized incremental algorithm [6, 18]. This algorithm actually computes the vertical decomposition $\mathcal{A}^{\|}(\mathcal{C}^*)$ of the arrangement, together with a point-location data structure with $O(\log n)$ expected query time. We use this data structure to locate the points ℓ^* dual to all lines $\ell \in S \setminus R$. From this we can determine, for every ℓ, the regions of $\mathcal{C}^*$ containing ℓ^*, or in other words, the polygons of $\mathcal{C}$ intersecting ℓ. Indeed, after having located all points of the form ℓ^*, we traverse the adjacency graph of the trapezoids in $\mathcal{A}^{\|}(\mathcal{C}^*)$. At each trapezoid $\tau \in \mathcal{A}^{\|}(\mathcal{C}^*)$ we compute $\mathcal{C}^*(\tau)$, the set of regions that contain the trapezoid $\tau \in \mathcal{A}^{\|}(\mathcal{C}^*)$, and output the pairs (ℓ, C) for $\ell^* \in \tau$ and $C^* \in \mathcal{C}^*(\tau)$. Suppose we arrive at τ from τ'; then $\mathcal{C}^*(\tau)$ and $\mathcal{C}^*(\tau')$ differ by at most one region (the region whose boundary separates τ from τ'), and thus $\mathcal{C}^*(\tau)$ can be obtained from $\mathcal{C}^*(\tau')$ in $O(1)$ time.

The total time spent in this step is $O((m^2+n)\log n)$ plus the number of polygon/line incidences. The expected number of these incidences is bounded by $O(\kappa(r,m)\cdot(n/r)) = O(m^2+n)$, using Proposition 2.1 with $r = n/t$ and $c = 1$.

Step 5. Let C be a cell in $\mathcal{A}(R,P)$, and let $S_C \subseteq S \setminus R$ be the set of lines intersecting the interior of C. For each line $\ell \in S_C$, we compute the trapezoids of $C^{\|}$ intersected by ℓ, as follows. We compute, in $O(\log n)$ time, the intersection points of ∂C and ℓ and also the trapezoids of $C^{\|}$ containing these intersection points. Next, by tracing ℓ through $C^{\|}$, we compute all b trapezoids of $C^{\|}$ that ℓ intersects. The time spent in finding the intersection points and tracing ℓ is $O(\log n + b)$. By repeating this procedure for all cells C and all lines $\ell \in L \setminus R$, we obtain S_Δ, for every $\Delta \in \mathcal{A}^{\|}(R,P)$. The running time of this step is $O(\sum_{\Delta\in\mathcal{A}^{\|}(R,P)} w(\Delta)\log n)$. The expected value of $O(\sum_{\Delta\in\mathcal{A}^{\|}(R,P)} w(\Delta))$, by Proposition 2.1 as before, is $O(m^2+n)$. Hence, the expected time spent in step 5 is $O((m^2+n)\log n)$.

Step 6. Let Δ be a trapezoid of $\mathcal{A}^{\|}(R,P)$. After having computed S_Δ, we compute the arrangement $\mathcal{A}(S_\Delta)$ using, say, a randomized incremental algorithm. We clip $\mathcal{A}(S_\Delta)$ within Δ and compute the vertical decomposition of the clipped arrangement. For each point $p \in P \cap \Delta$, we also compute the trapezoid of this vertical decomposition containing p. The time spent in this step is easily seen to be $O(w(\Delta)^2 + |P\cap\Delta|\log w(\Delta))$ per trapezoid $\Delta \in \mathcal{A}^{\|}(R,P)$.

For a cell $C \in \mathcal{A}(R,P)$, let $\mathbf{\Delta}_C$ be the set of resulting trapezoids that lie in C. We now define a graph $\mathcal{G}_C$ on the trapezoids of $\mathbf{\Delta}_C$. The vertices of $\mathcal{G}_C$ are the trapezoids of $\mathbf{\Delta}_C$, and two trapezoids are connected by an edge if they share a vertical edge. By performing a depth-first search on $\mathcal{G}_C$, we can extract all connected components of $\mathcal{G}_C$ whose trapezoids contain any point of P. That is, we pick a point $p \in P \cap C$. Let $\tau_p \in \mathbf{\Delta}_C$ be the trapezoid containing p. We perform a depth first search in $\mathcal{G}_C$ starting from τ_p until we find the entire connected component of $\mathcal{G}_C$ containing τ_p. Let $\mathbf{\Delta}_C(p)$ be the set of trapezoids in this component; then the union of these trapezoids is exactly the cell of $\mathcal{A}(S,\{p\})$. The vertices of the cell, sorted in clockwise order, can be obtained by merging the trapezoids of $\mathbf{\Delta}_C(p)$ in an obvious manner.

If there is a point $q \in P \cap C$ that does not lie in $\mathbf{\Delta}_C(p)$, we repeat the same procedure with q. We continue this process until we have extracted all components of $\mathcal{G}_C$ that contain any point of $P \cap C$. This gives $\mathcal{A}(S, P\cap C)$.

Repeating this step for all cells of $\mathcal{A}(R,P)$, we obtain all cells of $\mathcal{A}(S,P)$. Finally, we compute the vertical decomposition of all the cells. The total running time for Step 6 is

$$O(m\log n) + \sum_{\Delta\in\mathcal{A}(R,P)} O(w(\Delta)^2)\,,$$

and its expected value is

$$O(m \log n + \kappa(r, m)(n/r)^2) = O(m^2 + n)\,.$$

Putting all the pieces together, the total expected running time of Steps 4–6 is $O((m^2 + n) \log n)$. Let $T(n, m)$ denote the maximum expected time of the entire algorithm; then we obtain the following recurrence.

$$T(n, m) \leq \begin{cases} c_1 & \text{if } n \leq n_0, \\ \displaystyle\sum_{i=1}^{q} T\left(\lfloor n/t \rfloor, m_i\right) + c_2(m^2 + n) \log n & \text{if } n > n_0, \end{cases}$$

where $m_i \leq m/\sqrt{t}$ for $i \leq q = \lceil \sqrt{t}\, \rceil$, $\sum_{i=1}^{q} m_i = m$, and c_1, c_2 are appropriate constants. The solution of this recurrence is

$$T(n, m) = O((m^2 + n) \log n)\,.$$

If $m > \sqrt{n}$, we can divide the points of P into groups of size $\sqrt{n}$ and solve the subproblems separately. This standard batching technique yields a more convenient bound for the expected running time, namely $O((m\sqrt{n} + n) \log n)$. Hence, we can conclude the following lemma.

LEMMA 3.1. *Given a set S of n lines and a set P of $m \leq n^2$ points in the plane, the cells of $\mathcal{A}(S)$ containing the points of P can be computed by a randomized algorithm in expected time $O((m\sqrt{n} + n) \log n)$.*

We now present another randomized algorithm whose running time is significantly better for larger values of m. Although the basic idea is the same as in [1], the algorithm presented here is simpler because we allow randomization.

We choose a random subset $R \subseteq S$ of size r, where $r = \lceil m^{2/3}/n^{1/3} \rceil$. Using a randomized incremental algorithm, we construct $\mathcal{A}^{\shortparallel}(R)$ plus a point-location data structure for $\mathcal{A}^{\shortparallel}(R)$ in expected time $O(r^2)$ [6]. For each trapezoid $\Delta \in \mathcal{A}^{\shortparallel}(R)$, let $S_\Delta \subseteq S \setminus R$ be the set of lines that intersect the interior of Δ and $P_\Delta \subseteq P$ the set of points that are contained in Δ. S_Δ can be computed in time $O(nr)$ by tracing each line through $\mathcal{A}^{\shortparallel}(R)$ and P_Δ can be computed in expected time $O(m \log n)$ by locating each point of P in $\mathcal{A}^{\shortparallel}(R)$. Set $n_\Delta = |S_\Delta|$ and $m_\Delta = |P_\Delta|$. For the sake of convenience, we assume that $\mathcal{A}(S_\Delta), \mathcal{A}(S_\Delta, P_\Delta)$ are clipped within Δ, and $\mathcal{A}^{\shortparallel}(S_\Delta), \mathcal{A}^{\shortparallel}(S_\Delta, P_\Delta)$ are their vertical decompositions. Let Z_Δ denote the set of cells in $\mathcal{A}(S_\Delta)$ that intersect the vertical edges of Δ. It is well known that the number of edges in the faces in Z_Δ is $O(n_\Delta)$ [9].

Let p be a point of P_Δ. If the cell $\mathcal{A}(S_\Delta)$ containing p lies entirely in the interior of Δ, then $\mathcal{A}(S, \{p\}) = \mathcal{A}(S_\Delta, \{p\})$. Otherwise, $A(S, \{p\})$ may have edges that lie outside Δ, but each such edge lies on the boundary of faces in $Z_{\Delta'}$, $\Delta' \in A^{\shortparallel}(R)$. Hence, for each trapezoid Δ, it is sufficient to compute $\mathcal{A}(S_\Delta, P_\Delta)$ and Z_Δ. We compute $\mathcal{A}(S_\Delta, P_\Delta)$ in expected time $O((m_\Delta\sqrt{n_\Delta} + n_\Delta) \log n_\Delta)$ using Lemma 3.1. If we clip the lines of S_Δ within Δ, then Z_Δ is the unbounded face in the arrangement of the clipped segments, and we can compute it in expected time $O(n_\Delta \log n_\Delta)$ using a (simplified version of) the algorithm by Chazelle et al. [6]. Hence, the expected running time of the algorithm is

$$\mathrm{E}\left[\sum_{\Delta \in \mathcal{A}^{\shortparallel}(R)} O\left((m_\Delta\sqrt{n_\Delta} + n_\Delta) \log n_\Delta\right)\right] + O(nr) + O(m \log n)\,.$$

By a result of Clarkson and Shor [10, Theorem 3.6], we have

$$\mathrm{E}\left[\sum_{\Delta\in\mathcal{A}^{\shortparallel}(R)} n_\Delta \log n_\Delta\right] = O\left(nr\log\frac{n}{r}\right) \quad \text{and}$$

$$\mathrm{E}\left[\sum_{\Delta\in\mathcal{A}^{\shortparallel}(R)} m_\Delta\sqrt{n_\Delta}\log n_\Delta\right] = O\left(m\sqrt{\frac{n}{r}}\log\frac{n}{r}\right).$$

Thus, the expected running time of the algorithm is bounded by

$$O\left(\left(m\sqrt{\frac{n}{r}}+nr\right)\log\frac{n}{r}+m\log n\right).$$

Substituting the value r in the above expression, we obtain the following theorem.

THEOREM 3.2. *Given a set S of n lines and a set P of m points in the plane, the faces of $\mathcal{A}(S)$ containing the points of P can be computed by a randomized algorithm in expected time*

$$O\left(m^{2/3}n^{2/3}\log\frac{n}{\sqrt{m}}+(m+n)\log n\right).$$

4. Computing cells in segment arrangements. Next, we present an algorithm for computing marked cells in arrangements of segments. Let S be a set of n segments and P a set of m points in the plane. The goal is to compute $\mathcal{A}(S,P)$ and its vertical decomposition $\mathcal{A}^{\shortparallel}(S,P)$. Again, we begin with a simpler algorithm, which is effective for computing few cells, and then plug in the random-sampling technique to handle larger values of m.

The outline of the first algorithm is the same as in the previous section except that we must now interpret the operations in terms of segments. Since the cells of $\mathcal{A}^{\shortparallel}(R,P)$ are not necessarily simply connected, the boundary of m cells may consist of $m+n$ polygonal chains. Consequently, the computation of the sets of cells intersected by each segment of $S\setminus R$ in Step 4 and the computation of S_Δ for each trapezoid $\Delta\in\mathcal{A}^{\shortparallel}(R,P)$ in Step 5 now become considerably more complicated. Another difficulty in computing S_Δ is that we now have to detect intersections between simple polygons and segments rather than between convex polygons and lines. In the remainder of this section we will describe the details of Steps 4 and 5.

The boundary ∂C of each cell $C\in\mathcal{A}(R,P)$ is composed of (at most) one *outer* component and a family of *inner* components; C lies in the interior of the outer component and in the exterior of each inner component. Each component of ∂C can be regarded as a simple polygonal chain. Let $\mathcal{O}$ be the set of outer boundary components of the cells in $\mathcal{A}(R,P)$, and let $\mathcal{I}$ be the set of the inner boundary components of these cells. We have $|\mathcal{O}|\le m$ and $|\mathcal{I}|\le m+n$. Let μ be the total number of edges of all polygons in $\mathcal{O}\cup\mathcal{I}$; obviously,

$$\mu\le\eta(n/t,m)=O(m^2\log m+n\log m+n\alpha(n)); \tag{4.1}$$

the last inequality follows from a weaker result of Aronov et al. [2].

We first decompose each segment $g\in S\setminus R$ into maximal subsegments so that each subsegment lies in the interior of some outer component O. We cut each segment at the intersection points of $\mathcal{O}$ and S and discard the subsegments that lie in the exterior

of $\mathcal{O}$. Let Σ be the set of resulting subsegments. Next, for each subsegment $\sigma \in \Sigma$, we compute the trapezoids of $\mathcal{A}^{\parallel}(R, P)$ intersected by σ.

Suppose that we have already computed Σ in Step 4. Then in Step 5 we compute S_Δ, for all $\Delta \in \mathcal{A}^{\parallel}(R, P)$, as follows. We preprocess each polygonal chain $I \in \mathcal{I}$, in linear time, for ray-shooting queries, so that the first intersection point of a query ray and I can be computed in logarithmic time; see [5, 15]. The total time spent in preprocessing $\mathcal{I}$ is $O(\mu) = O(\eta(n, m))$.

Let σ be a subsegment of Σ that lies in the interior of the outer component of the boundary of a cell C. We trace σ through $C^{\parallel}$ to compute the trapezoids of $C^{\parallel}$ intersected by σ. In more detail, let a, b be the endpoints of σ, and let $\Delta(a)$ be the trapezoid of $C^{\parallel}$ containing a. If a is not an endpoint of a segment of $S \setminus R$, then a lies on the boundary of $\Delta(a)$. We check whether $b \in \Delta(a)$. If the answer is "yes," then $\Delta(a)$ is the only trapezoid of $C^{\parallel}$ intersected by σ, and we stop. If $b \notin \Delta(a)$, we compute the other intersection point, a_1, of σ and $\Delta(a)$. If a_1 lies on a vertical edge of $\Delta(a)$, we also compute, in constant time, the next trapezoid $\Delta(a_1)$ of $C^{\parallel}$ intersected by σ and repeat the same step with a_1 and $\Delta(a_1)$. If a_1, on the other hand, lies on an edge of the cell C, then a_1 lies on the boundary of some inner component $I \in \mathcal{I}$ of C, and the portion of the segment σ immediately following a_1 lies outside C. Using the ray-shooting data structure, we compute the next intersection point a_2 of the polygonal chain I and the segment $\overrightarrow{a_1 b}$. Once we know a_2, we can also compute the trapezoid of $C^{\parallel}$ containing a_2, and we continue tracing σ through $C^{\parallel}$.

For each trapezoid intersected by σ, we spend $O(\log n)$ time, so the total time spent in computing the k_σ trapezoids intersected by σ is $O(k_\sigma \log n)$. Summing over all segments of Σ, the total time spent is

$$\sum_{\sigma \in \Sigma} O(k_\sigma \log n) = O\left(\sum_{\Delta \in \mathcal{A}^{\parallel}(R,P)} n_\Delta \log n \right),$$

where $n_\Delta = |S_\Delta|$.

Next, we describe how to compute the set Σ. Notice that it is sufficient to compute all intersection points between the segments of $S \setminus R$ and the outer polygonal chains in $\mathcal{O}$.

Let $\mathcal{J}$ be the set of intervals corresponding to the x-projections of the polygonal chains in $\mathcal{O}$. We construct in time $O(m \log m)$ an interval tree T on $\mathcal{J}$; see [17] for details on interval trees. T is a minimum-height binary tree with at most $2m$ leaves, each of whose nodes v is associated with a vertical strip W_v and a point x_v; x_v is the median of the endpoints of $\mathcal{J}_v$ that lie in the interior of W_v. For the root u, W_u is the entire plane. If v, z are the children of u, then W_v and W_z are obtained by splitting W_u into two vertical strips by the vertical line passing through x_u. An interval $J \in \mathcal{J}$ is stored at the highest node v of T for which $x_v \in J$.

Let $\mathcal{O}_v \subseteq \mathcal{O}$ be the subset of polygonal chains whose projections are stored at v. Each chain belongs to exactly one $\mathcal{O}_v$. Let $Z_v = \bigcup_w \mathcal{O}_w$, where the union is taken over all descendants of v, including v; set $z_v = |Z_v|$. Finally, let ζ_v denote the total number of edges in Z_v. Since each polygonal chain of $\mathcal{O}$ appears in at most $O(\log m)$ Z_v's, we have $\sum_{v \in T} \zeta_v = O(\mu \log m)$, where μ is, as above, the total number of edges in $\mathcal{O}$. Moreover, by the construction of T, if v_1, v_2 are the children of v, then $z_{v_1}, z_{v_2} \leq z_v/2$, and therefore $\sum_{v \in T} z_v^2 = O(m^2)$.

The polygonal chains in $\mathcal{O}_v$ are pairwise disjoint and all of them intersect the vertical line passing through x_v, so we can regard $\mathcal{O}_v$ along with appropriate portions

of the vertical line as a simple polygon and preprocess it in linear time for answering ray-shooting queries. The time spent in this step is $O(\mu)$, as each polygonal chain belongs to exactly one O_v. Using this data structure, all a intersection points of a segment g and $\mathcal{O}_v$ can be reported in time $O((a+1)\log n)$.

Next, we take the convex hull of each polygonal chain in Z_v and preprocess the resulting convex polygons into a data structure, as described in the previous section, so that all convex polygons intersected by a query line can be reported quickly. Since any two polygonal chains of $\mathcal{O}$ are disjoint, the boundaries of their convex hulls intersect in at most two points and they have at most four common tangents. Consequently, the line intersection-searching structure has size $O(z_v^2+\zeta_v)$, and it can be computed in time $O(z_v^2 + z_v \log \zeta_v + \zeta_v)$, using the algorithm described in [19]. We also preprocess each $O \in \mathcal{O}$ in linear time for ray-shooting queries as in [15]. The total time spent in preprocessing $\mathcal{O}$ is therefore

$$\begin{aligned} &O\left(m\log m + \mu + \sum_v \left(z_v^2 + z_v \log \zeta_v + \zeta_v\right)\right) \\ &\qquad = O(m^2 + m\log m \log n + \mu \log m) \\ &\qquad = O((m^2 \log m + n \log m + n\alpha(n)) \log m), \end{aligned} \tag{4.2}$$

where the last inequality follows from (4.1).

Let $g \in S \setminus R$ be a segment. All intersection points of g and $\mathcal{O}$ can be computed as follows. We search the tree T with g starting from the root. Let v be a node visited by the query procedure. We need to compute the intersection points of g and the polygonal chains in Z_v. If the endpoints of g do not lie in the vertical strip W_v, that is, g completely crosses W_v, then g intersects a polygonal chain $O \in Z_v$ if and only if the line supporting g intersects the convex hull of O. Thus, we first compute all polygonal chains of Z_v intersected by the line supporting g, using the line intersection-searching structure, and then, for each $O \in Z_v$ intersected by g, we compute the intersection points of g and O using the ray-shooting data structure. If a is the number of intersection points between g and the polygonal chains of Z_v, then the total time in reporting these intersections is $O((a+1)\log n)$. If, on the other hand, one of the endpoints of g lies in W_v, we can compute all b intersection points between $\mathcal{O}_v$ and g in time $O((b+1)\log n)$ using the ray-shooting data structure for $\mathcal{O}_v$. Let v_1, v_2 be the children of the node v. If g intersects W_{v_1} (resp., W_{v_2}), we recursively visit v_1 (resp., v_2).

It is easily seen that the query procedure visits $O(\log m)$ nodes and that the query time is $O((\log m + k_g)\log n)$, where k_g is the total number of intersection points reported. We repeat this procedure for all segments $g \in S \setminus R$. Since

$$\sum_{g \in S\setminus R} k_g \;\le\; \sum_{\Delta \in \mathcal{A}^{\shortparallel}(R,P)} n_\Delta\,,$$

the total cost of computing the intersection points between $S \setminus R$ and $\mathcal{O}$ is

$$\sum_{g\in S\setminus R} O((\log m + k_g)\log n) = O\left(n \log m \log n + \sum_{\Delta \in \mathcal{A}^{\shortparallel}(R,P)} n_\Delta \log n\right).$$

The expected value of $\sum_\Delta n_\Delta$ is $\eta(r,m)O(n/r) = O(\eta(r,m))$; therefore we obtain

$$\sum_{g\in S\setminus R} O((\log m + k_g)\log n) = O(n\log m \log n + \eta(r,m)\log n). \tag{4.3}$$

As in the previous section, the time spent in step 6 (refining the cells of $\mathcal{A}^{\shortparallel}(R,P)$) is $O(\sum_\Delta n_\Delta^2)$. Using Proposition 2.1 (ii), we obtain that

$$\mathrm{E}\left[\sum_\Delta n_\Delta^2\right] = \eta(r,m)O((n/r)^2) = O(\eta(r,m)). \tag{4.4}$$

Summing (4.2), (4.3), and (4.4), and using the fact that $m \le n^2$, the total expected time spent in the merge step is $O((m^2\log m + n\log m + n\,\alpha(n))\log n)$.

This gives the following recurrence for $T(n,m)$ (the maximum expected running time):

$$T(n,m) \le \begin{cases} c_1 \quad \text{if } n \le n_0, \\ \displaystyle\sum_{i=1}^{\sqrt{t}} T\left(\frac{n}{t}, m_i\right) + O((m^2\log m + n\log m + n\,\alpha(n))\log n) \quad \text{if } n > n_0, \end{cases}$$

where $m_i \le m/\sqrt{t}$ for all $i \le \sqrt{t}$. The solution of this recurrence is

$$T(n,m) = O((m^2\log m + n\log m + n\alpha(n))\log n).$$

Hence, we can conclude that the expected running time of the overall algorithm for computing $\mathcal{A}(S,P)$ is

$$O\left((m^2\log m + n\log m + n\,\alpha(n))\log n\right).$$

We can again use the same batching technique if m is large. That is, partition P into groups of size $\sqrt{n}$ and solve the subproblems separately. Omitting the details, we obtain the following lemma.

LEMMA 4.1. *Given a set S of n segments and a set P of m points in the plane, the faces of $\mathcal{A}(S)$ that contain at least one point of P can be computed by a randomized algorithm in expected time $O\left((m\sqrt{n}\log n + n\log m + n\alpha(n))\log n\right)$.*

For larger values of m, we again use the random-sampling technique as in the previous section. That is, we choose a random subset $R \subseteq S$ of size $r = \lceil m^{2/3}/n^{1/3}\rceil$ and compute $\mathcal{A}^{\shortparallel}(R)$. For each $\Delta \in \mathcal{A}^{\shortparallel}(R)$, we compute $P_\Delta = P \cap \Delta$ and S_Δ, the set of segments that intersect Δ. We clip the segments within Δ. The total expected time spent in this step is $O(r^2 + (m+nr)\log r)$. Let z be a point lying in the unbounded face of $\mathcal{A}(S)$. For each $\Delta \in \mathcal{A}^{\shortparallel}(R)$, we compute $\mathcal{A}^{\shortparallel}(S_\Delta, P_\Delta \cup \{z\})$, in expected time

$$O((m_\Delta\sqrt{n_\Delta}\log m_\Delta + n_\Delta\log m_\Delta + n_\Delta\alpha(n_\Delta))\log n_\Delta),$$

using Lemma 4.1, and then glue these faces together. The overall expected running time of the algorithm is

$$\mathrm{E}\left[\sum_{\Delta\in\mathcal{A}^{\shortparallel}(R)} O((m_\Delta\sqrt{n_\Delta}\log n_\Delta + n_\Delta\alpha(n_\Delta) + n_\Delta\log m_\Delta)\log n_\Delta)\right] + O((m+nr)\log r)\,. \tag{4.5}$$

Again, using the results by Clarkson–Shor [10], we have

$$\mathrm{E}\left[\sum_{\Delta\in\mathcal{A}^{\shortparallel}(R)} m_\Delta\sqrt{n_\Delta}\log n_\Delta\right] = O\left(m\sqrt{\frac{n}{r}}\log\frac{n}{r}\right) \tag{4.6}$$

$$\sum_{\Delta\in\mathcal{A}^{\shortparallel}(R)} n_\Delta = \mathrm{E}\left[|\mathcal{A}^{\shortparallel}(R)|\right]O\left(\frac{n}{r}\right) = O(nr). \tag{4.7}$$

Substituting (4.5), (4.6), and the value of r in (4.7), we obtain the following theorem.

THEOREM 4.2. *Given a set S of n segments and a set P of m points in the plane, the faces of $\mathcal{A}(S)$ that contain a point of P can be computed by a randomized algorithm in expected time*

$$O\left(m^{2/3}n^{2/3}\log^2\frac{n}{\sqrt{m}} + (m + n\log m + n\,\alpha(n))\log n\right).$$

Finally, we remark that if $\mathcal{A}(S)$ has only $k = o(n^2)$ vertices, then using the fact that the expected number of trapezoids in $\mathcal{A}^{\|}(R)$ is $O(kr^2/n^2 + r)$ we can obtain a better bound on the expected running time of the algorithm. In particular, choosing $r = \lceil n(m/k)^{2/3}\rceil$ and using (4.4), (4.5), it can be shown that the expected running time of the algorithm is

$$O\left(m^{2/3}k^{1/3}\log^2\frac{k}{m} + (m + n\log m + n\,\alpha(n))\log n\right).$$

Acknowledgments. The authors thank Mark de Berg, Mark Overmars, and Micha Sharir for several useful discussions. The comments by an anonymous referee helped improve the presentation of the paper.

REFERENCES

[1] P. K. AGARWAL, *Partitioning arrangements of lines:* II. *Applications*, Discrete Comput. Geom., 5 (1990), pp. 533–573.
[2] B. ARONOV, H. EDELSBRUNNER, L. GUIBAS, AND M. SHARIR, *Improved bounds on the complexity of many faces in arrangements of segments*, Combinatorica, 12 (1992), pp. 261–274.
[3] M. DE BERG, K. DOBRINDT, AND O. SCHWARZKOPF, *On lazy randomized incremental construction*, Discrete Comput. Geom., 14 (1995), pp. 261–286.
[4] R. CANHAM, *A theorem on arrangements of lines in the plane*, Israel J. Math., 7 (1969), pp. 393–397.
[5] B. CHAZELLE, H. EDELSBRUNNER, M. GRIGNI, L. GUIBAS, J. HERSHBERGER, M. SHARIR, AND J. SNOEYINK, *Ray shooting in polygons using geodesic triangulations*, Algorithmica, 12 (1994), pp. 54–68.
[6] B. CHAZELLE, H. EDELSBRUNNER, L. GUIBAS, M. SHARIR, AND J. SNOEYINK, *Computing a face in an arrangement of line segments*, SIAM J. Comput., 22 (1993), pp. 1286–1302.
[7] B. CHAZELLE AND J. FRIEDMAN, *A deterministic view of random sampling and its use in geometry*, Combinatorica, 10 (1990), pp. 229–249.
[8] K. CLARKSON, *Computing a single face in an arrangement of segments*, 1990, manuscript.
[9] K. CLARKSON, H. EDELSBRUNNER, L. GUIBAS, M. SHARIR, AND E. WELZL, *Combinatorial complexity bounds for arrangements of curves and spheres*, Discrete Comput. Geom., 5 (1990), pp. 99–160.
[10] K. CLARKSON AND P. SHOR, *Applications of random sampling in computational geometry* II, Discrete Comput. Geom., 4 (1989), pp. 387–421.
[11] H. EDELSBRUNNER, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin, 1987.
[12] H. EDELSBRUNNER, L. GUIBAS, AND M. SHARIR, *The complexity of many faces in arrangements of lines and of segments*, Discrete Comput. Geom., 5 (1990), pp. 161–196.
[13] H. EDELSBRUNNER AND E. WELZL, *On the maximal number of edges of many faces in an arrangement*, J. Combin. Theory Ser. A, 41 (1986), pp. 159–166.
[14] L. GUIBAS AND M. SHARIR, *Combinatorics and algorithms of arrangements*, in New Trends in Discrete and Computational Geometry, J. Pach, ed., Springer-Verlag, Heidelberg, 1993, pp. 9–36.
[15] J. HERSHBERGER AND S. SURI, *A pedestrian approach to ray shooting: Shoot a ray, take a walk*, J. Algorithms, 18 (1995), pp. 403–431.
[16] D. HAUSSLER AND E. WELZL, *ϵ-nets and simplex range queries*, Discrete Comput. Geom., 2 (1987), pp. 127–151.
[17] K. MEHLHORN, *Data Structures and Algorithms* 3*: Multi-dimensional Searching and Computational Geometry*, Springer-Verlag, Berlin, 1984.
[18] K. MULMULEY, *A fast planar partition algorithm*, I, J. Symbolic Comput., 10 (1990), pp. 253–280.

[19] K. Mulmuley, *A fast planar partition algorithm*, II, J. Assoc. Comput. Mach., 38 (1991), pp. 74–103.
[20] M. Sharir and P. K. Agarwal, *Davenport-Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, New York, 1995.
[21] E. Szemerédi and W. Trotter Jr., *Extremal problems in discrete geometry*, Combinatorica, 3 (1983), pp. 381–392.

SIAM J. COMPUT.
Vol. 27, No. 2, pp. 506–544, April 1998
© 1998 Society for Industrial and Applied Mathematics
010

FAULT-TOLERANT COMPUTATION IN THE FULL INFORMATION MODEL*

ODED GOLDREICH†, SHAFI GOLDWASSER‡, AND NATHAN LINIAL§

Abstract. We initiate an investigation of general fault-tolerant distributed computation in the *full-information* model. In the full information model no restrictions are made on the computational power of the faulty parties or the information available to them. (Namely, the faulty players may be infinitely powerful and there are no private channels connecting pairs of honest players).

Previous work in this model has concentrated on the particular problem of simulating a single bounded-bias global coin flip (e.g., Ben-Or and Linial [*Randomness and Computation*, S. Micali, ed., JAI Press, Greenwich, CT, 1989, pp. 91–115] and Alon and Naor [*SIAM J. Comput.*, 22 (1993), pp. 403–417]). We widen the scope of investigation to the general question of how well arbitrary fault-tolerant computations can be performed in this model. The results we obtain should be considered as first steps in this direction.

We present efficient two-party protocols for fault-tolerant computation of any bivariate function. We prove that the advantage of a dishonest player in these protocols is the minimum one possible (up to polylogarithmic factors).

We also present efficient m-party fault-tolerant protocols for sampling a general distribution ($m \geq 2$). Such an algorithm seems an important building block towards the design of efficient multiparty protocols for fault-tolerant computation of multivariate functions.

Key words. fault-tolerant multiparty protocols, influences in general two-party computations, sampling with weak sources of randomness

AMS subject classifications. 68Q10, 68Q22, 68Q75

PII. S0097539793246689

1. Introduction.

The problem of how to perform general distributed computation in an unreliable environment has been extensively addressed. Two types of models have been considered. The first model assumes that one-way functions exist and considers adversaries (faults) which are computationally restricted to probabilistic polynomial time [24, 13, 25, 14, 11, 2]. The second model postulates that private channels exist between every pair of players [3, 7, 8, 17, 15]. Hence, in both models fault-tolerance is achieved at the cost of restricting the type of faults.

We want to avoid any such assumption and examine the problem of fault-tolerant distributed computation where the faults are computationally unrestricted, and no private channels are available. Clearly, the assumption that one-way functions exist is of no use here. The situation here corresponds to games of complete information.

The general problem can be described informally as follows: m players are interested in globally computing $v = f(x_1, \ldots, x_m)$, where f is a predetermined m-variate function and x_i is an input given to party i (and initially known only to it). The input x_i is assumed to have been drawn from probability distribution D_i (which without loss of generality can be assumed to be uniform). A coalition F of faulty players may

*Received by the editors April 2, 1993; accepted for publication (in revised form) March 14, 1996. This research was supported by grants 89-00312, 89-00126, and 92-00226 from the United States–Israel Binational Science Foundation (BSF), Jerusalem, Israel. An extended abstract of this work has appeared in the *Proceedings of the* 32*nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, IEEE Computer Society Press, Los Alamitos, CA, 1991, pp. 447–457.

http://www.siam.org/journals/sicomp/27-2/24668.html

†Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel (oded@wisdom.weizmann.ac.il).

‡Laboratory for Computer Science, MIT, Cambridge, MA 02139 (shafi@theory.les.mit.edu).

§Institute of Computer Science, Hebrew University, Jerusalem, Israel (nati@humus.huji.ac.il).

favor a particular value v for f and play any strategy to maximize the probability of such an outcome. We want to bound, for each value v in the range of f, the probability (under the best strategy for the faults) that the outcome of the protocol used to distributively compute f is v. How good can this bound be?

Regardless of the protocol under consideration, there is always one avenue that is open for the faulty players, namely, alter their input values to ones under which the value v is most likely. This is always possible, since players' inputs are not visible to others. That is,

$$q_v := \max_{x_i, i \in F} \{\text{Prob}(f(\vec{x}) = v \text{ where } x_j \in_R D_j, j \notin F)\}$$

is a lower bound on the influence of coalition F towards value v, no matter what protocol is used.

Consider the simple procedure in which each player announces its x_i, and the global output is taken to be $f(x_1, \ldots, x_m)$. If all players (including the faulty ones) act simultaneously, then for every v, the probability of v being the outcome is indeed at most q_v. Unfortunately, in a distributed network simultaneity cannot be guaranteed, and a delayed action by the faults can result in much better performance for them (e.g., for $f = \sum_{i=1}^{m} x_i \bmod N$ with $x_i \in \{0, 1, \ldots, N-1\}$, $q_0 = \frac{1}{N}$, but a single faulty player acting last has complete control of the outcome).

In both of the previously studied models (private channels or computationally bounded faults) protocols were developed where for all values v and all *minority* coalitions F, the probability of outcome v is as close to q_v as desired. The key to these protocols is the notion of *simultaneous commitment*. At the outset of these protocols, each player P_i commits to its input x_i. It should be stressed that a faulty party may alter its input in this "committing phase" but not later and that a party's commitment is "independent" of the inputs of the other honest parties.

Obviously, in the full-information model such a qualitative notion of commitment cannot be implemented (even if the faulty parties are in minority). Instead, we need to look for *quantitative* results. Faulty players can and will be able to "alter their inputs" throughout the execution of the protocol in order to influence the outcome. Yet, we can bound the advantage gained by their improper behavior.

1.1. Results concerning the two-party case. The main focus of this paper is on the two-player case of this problem. Even this restricted case provides interesting problems and challenges. We resolve the main problems in this case, showing:

1. A lower bound: for every bivariate function f, for any protocol to compute f, and every value v in the range of f, there is a strategy for one of the players, so that if the other player plays honestly, then the probability for the outcome $f = v$ is at least $\max(q_v, \sqrt{p_v})$, where $p_v = \text{Prob}(f(\vec{x}) = v | x_i \in_R D_i))$.
2. More interestingly, we show a matching (up to polylogarithmic factor) constructive upper bound. We describe a probabilistic polynomial-time protocol that computes f, given a single oracle access to f, such that for all v,

$$Pr(f \text{ evaluates to } v) = O(\text{poly}\log(1/p_v) \cdot \max(q_v, \sqrt{p_v})).$$

   In the special case where $q_v = p_v$, this protocol is shown to match the lower bound up to a constant factor. Namely,

$$Pr(f \text{ evaluates to } v) = O(\sqrt{p_v}).$$

The spirit of our protocol is best illustrated by the following example.

Example. Define $id(x,y)=1$ if $x=y$ and 0 otherwise. Suppose that the local inputs x,y are chosen uniformly in $\{0,1\}^n$. Clearly, $p_1=\frac{1}{N}$, and $p_0=1-\frac{1}{N}$, where $N=2^n$. A protocol in which the first player declares x and then the second player declares y allows the second player complete control on the value of id. A protocol in which the two players alternately exchange bits in the description of their inputs is no better if these bits are exchanged in the same order (i.e., both parties send their respective ith bit in round i). A much better idea is for the two players to alternate in describing the bits of their inputs but do so from opposite directions (i.e., in round i the first party sends its ith bit, whereas the second party sends its $(n-i+1)$st bit). Clearly, whichever player is faulty, the probability that the outcome of this protocol is "1" is bounded by $\frac{1}{\sqrt{N}}$. In light of the lower bound, this is the best result possible. This idea of gradually revealing appropriately chosen "bits of information" is the key to the general problem of two-party computation.

1.2. Results concerning the multiparty case. The problem of m-party computations, where a subset of $t<m$ faults may exist, is more involved than the two-party case (even for $m=3$); see discussion in section 5. Here, we only consider the problem of collectively sampling a given distribution. Without loss of generality, it suffices to consider the uniform distribution (say, on strings in $\{0,1\}^l$). We provide a probabilistic polynomial-time sampling protocol such that for every $S\subset\{0,1\}^l$, for every t faults,

$$\Pr(\text{sample } \in S) < \left(\frac{|S|}{2^l}\right)^{1-c\cdot\frac{t}{m}}$$

for some constant $c>0$. This result is the best possible (up to the constant c), and is superior to the bound obtained by the trivial protocol which consists of l repeated applications of "collective coin flipping"; consider, for example, the set S consisting of all strings having at least $(\frac{1}{2}+\frac{t}{m})\cdot l$ ones; under the trivial protocol, t faulty parties can influence the output to almost always hit S, whereas our result guarantees that this set S which forms a negligible fraction of $\{0,1\}^l$ is hit with negligible probability (for, say, $t<m/2c$).[1]

The above sampling protocol can be used to present a (generic probabilistic polynomial-time) protocol that works well for computing *almost all* functions (see our technical report [12]).

1.3. Previous work in the full-information model. Previous work in this model [4, 5, 16, 1] has focused on the task known as *collective coin flipping*, which in our terminology amounts to fault-tolerant multiparty sampling in $\{0,1\}$. Matching lower and (constructive) upper bounds of $\frac{1}{2}+\theta(\frac{t}{m})$ have been shown (by Ben-Or and Linial [4] and Alon and Naor [1],[2] respectively). Our work can be viewed as an extension of these investigations which were concerned with the influences of players on *Boolean* functions (i.e., $Range(f)=\{0,1\}$). The general case considered in this paper gives rise to additional difficulties. Let us stress that even the problem of sampling in arbitrary sets is more difficult than collective coin flipping. As mentioned above, the obvious approach to the sampling problem fails; namely, a sampling protocol that

[1] Using the above choice of parameters, we have a set S of density $\rho\approx\exp\{-(t/m)^2\cdot l\}$ which our protocol hits with probability at most $\sqrt{\rho}$, as long as at most t players are faulty. On the other hand, when repeated collective coin flippings are used, t faulty players can influence the outcome to be in S with probability at least $1-\rho$, by biasing each coin flip toward 1.

[2] Furthermore, the upper bound can be met by protocols of logarithmic round-complexity [9, 19].

consists of repeatedly applying a given coin-tossing protocol can be easily influenced to almost always output strings in a subset of negligible size.[3]

However, fault-tolerant computation (of arbitrary functions) is more complex than sampling, which can be viewed as fault-tolerant computation of a function specially designed for this purpose.

1.4. Relation to work on slightly random sources. In this paper we present a multiparty protocol for sampling a set of strings $\{0,1\}^l$. In "sampling" we mean producing a single string in $\{0,1\}^l$ so that, for every subset $S \subset \{0,1\}^l$, the probability that the sample hits S is related to the density of S. Our protocol uses the collective coin flipping of [1] as a subroutine. In fact, our sampling protocol can be viewed as a deterministic reduction to the problem of collective coin tossing. The collective coin can be viewed as a slightly random source in the sense of Santha and Vazirani [22], i.e., an *SV-source*.[4] Hence, our result can be interpreted as presenting a sampling algorithm which uses an SV-source (with a parameter $\gamma < \frac{1}{\sqrt{2}}$). Our sampling algorithm performs much better than the obvious algorithm which uses as a sample a sequence of coins produced by the source. (The situation is analogous to the discussion of the multiparty sampling protocols above.)

Our sampling algorithm provides an alternative way of recognizing languages in BPP by polynomial-time algorithms which use an SV-source with a parameter $\gamma < \frac{1}{\sqrt{2}}$. First, reduce the error probability in the BPP-algorithm so that it is bounded by a sufficiently small constant. Next, use our sampling algorithm to produce a sequence of coin tosses for a *single run* of the new BPP-algorithm. Since the "bad runs" form a negligible fraction of all possible runs of the BPP-algorithm, it follows that the probability we will sample a bad run (when using an SV-source with parameter $\gamma < \frac{1}{\sqrt{2}}$) is bounded by $\frac{1}{3}$. This simulation method is different from the original method of Vazirani and Vazirani [23] (adopted also in [6]) where the BPP-algorithm is invoked *many* times, each time with a different sequence of coin tosses.

1.5. Other related work. We also present efficient sampling protocols for the two-party case. The basic sampling protocol guarantees, for every set $S \subseteq \{0,1\}^l$, that as long as one party is honest the output hits S with probability at most $O(\sqrt[4]{|S|/2^l})$. (The basic sampling protocol is essential for efficiently implementing our generic two-party function-computation protocol. Interestingly, the basic sampling protocol is also used as a building block for a better sampling protocol, which is optimal up to a constant factor.)

Our basic two-party sampling protocol is very similar to a protocol, called interactive hashing, which was discovered independently by Ostrovsky, Venkatesan, and Yung [20] (see Naor et al. [18]). Interactive hashing has found many applications in cryptography (cf. [20, 18, 21, 10]). For details see Remark 4.26.

1.6. Organization. We start with some preliminaries (section 2) and lower bounds (section 3). The main part of this paper is section 4, which presents efficient fault-tolerant two-party protocols. The construction of fault-tolerant multiparty protocols is discussed in section 5.

[3] An alternative method which also fails is to try to generalize the work of Alon and Naor [1] as follows: The method of [1] consists of randomly selecting one of the players who is appointed to flip a fair coin. Letting this player select a random string is a natural idea, but it is obvious that this approach performs very poorly for a sample space of nonconstant size. Specifically, each set $S \subset \{0,1\}^l$ can be hit with probability at least $\frac{t}{m}$, independently of S and l.

[4] An SV-source with parameter γ is a sequence of binary random variables $X_1, X_2, \ldots$, so that for every n, $\alpha \in \{0,1\}^n$ and $\sigma \in \{0,1\}$, $\text{Prob}(X_{n+1}=\sigma | X_1, \ldots, X_n = \alpha) \leq \gamma$.

2. Preliminaries. In this section, we present our conventions regarding functions and protocols. We also explain what we mean when we talk of influence and sampling.

2.1. Bivariate functions. Throughout the paper we represent the bivariate function $f:\{0,1\}^n\times\{0,1\}^n\mapsto\{0,1\}^*$ as an N-by-N matrix, where $N\stackrel{\text{def}}{=}2^n$. An entry, (x,y), in the matrix which has value v (i.e., $f(x,y)=v$) is called a *v-entry*. The following quantities, related to the function f and a value v in its range, are central to our analysis.

Notation. The *density of* v, denoted p_v, is the fraction of v-entries in the matrix of f (i.e., $p_v=|\{(x,y):f(x,y)=v\}|/2^{2n}$). The *maximum row density of* v, denoted r_v, is the maximum, taken over all rows, of the fraction of v-entries in a row of f (i.e., $r_v=\max_{x\in\{0,1\}^n}\{|\{y:f(x,y)=v\}|/2^n\}$). The *maximum column density of* v is denoted $c_v=\max_{y\in\{0,1\}^n}\{|\{x:f(x,y)=v\}|/2^n\}$, and q_v is defined as $\max\{r_v,c_v\}$.

Throughout the paper, we consider the case of uniform input distribution. Namely, we assume that each input is selected uniformly from $\{0,1\}^n$ and independently of the other input(s). The more general case, where each input is selected from an arbitrary distribution (yet independently of the other inputs) can be reduced to the uniform case as follows. Suppose that the probability for each input can be expressed as $\frac{q}{2^{\text{poly}(n)}}$, where q is an integer (for some polynomial poly). Then we can replace this input, say z, by q inputs, denoted $(z,1),(z,2),\ldots,(z,q)$, and consider the function $F((x,i),(y,j))\stackrel{\text{def}}{=}f(x,y)$ $(1\le i\le\phi(x)2^{\text{poly}(n)}$ and $1\le j\le\psi(y)2^{\text{poly}(n)}$, where $\phi(x)$ is the probability of the row-input x and $\psi(y)$ is the probability of the column-input y). Protocols for computing F (under the uniform distribution) translate easily to protocols for computing f (under the distribution (ϕ,ψ)) and vice versa. To efficiently transform protocols for computing F into protocols for computing f, an efficient algorithm is needed for computing the original density functions (i.e., ϕ and ψ).

2.2. Protocols. The communication model consists of a single broadcast channel. Each party can, at any time, place a message on this channel which arrives immediately (bearing the identity of its originator) to all other parties. It is not possible to impose "simultaneity" on the channel; namely, the protocols may not contain a mechanism ensuring simultaneous transmission of messages by different parties. Thus, it is best to think of the model as being asynchronous and of the protocols as being message-driven. However, asynchronicity is not a major issue here as all parties share the unique communication medium and thus have the same view.

The output of an execution of a protocol is defined as the last message sent during the execution. We consider the output of the protocol when the inputs are selected uniformly.

We call a player *honest* if it follows the protocol. *Dishonest* players may deviate arbitrarily from the protocol. In discussing our protocols we assume, without loss of generality, that dishonest players do not deviate from the protocol in a manner which may be detected. This assumption can be easily removed by augmenting our protocols with simple detection and recovery procedures (which determine the output of the protocol in case deviation from the protocol is detected). For example, the protocol may be restarted with the input of the cheating party fixed to some predetermined value and all its actions being simulated by the other parties.

All our protocols are *generic*: Players are instructed to take steps that depend only on their inputs, but not on the function f. When the inputs are finally revealed, f is evaluated once, and the protocol terminates.

2.3. Influences. Unlike previous work, we use the term "influence" in a colloquial manner. Typically, by talking "the influence of a party towards a value" we mean the probability that this party can make this value appear as output of the protocol. When discussing the computation of functions, we treat only the influence towards a single value; the influence towards a set of values can be treated by defining a corresponding indicator function.

2.4. Sampling. We also consider the problem of designing two-party and multiparty protocols for sampling in a universe $\{0,1\}^l$. The objective here is to provide upper bounds for the probability that the output falls in some subset $S \subset \{0,1\}^l$. We note that the problem of designing a two-party protocol for sampling $\{0,1\}^l$ can be reduced to the problem of designing a protocol for computing any function $f : \{0,1\}^n \times \{0,1\}^n \mapsto \{0,1\}^l$ for which all values have the same density and this density equals the maximum row/column densities (i.e., $q_v = p_v = 2^{-l}$ for every $v \in \{0,1\}^l$). An analogous reduction holds also in the multiparty case.

3. Lower bounds. In this section we present lower bounds which will guide our search for the best possible fault-tolerant protocols.

THEOREM 3.1. *Let $f : D_1 \times D_2 \times \cdots \times D_m \mapsto R$ be a function of m variables, Π an m-party protocol for computing f, and $v \in R$ a value in the range of f. Consider performing Π where players in the set S are dishonest, while all other players are honest. Let ϕ_S be the maximum, over all strategies of coalition S of the probability of the outcome being v. Then, for any $1 \le t \le m$ there is a coalition Q of t players with $\phi_Q \ge p_v^{1-\frac{t}{m}}$.*

In particular, we have the following result.

COROLLARY 3.2. *Let f be any bivariate function, Π any two-party protocol for computing f, and v a value in the range of f. Then at least one of the players can, by playing (possibly) dishonestly, force the outcome to be v with probability at least $\max\{q_v, \sqrt{p_v}\}$ (the other party plays honestly).*

Proof of Theorem 3.1. The proof is very similar to that of Theorem 5 in [4], although some changes are required. One observes first that if the time complexity of the protocol is no issue, and the only consideration is to keep influences down, then nothing is lost if all actions are taken sequentially and not in parallel. Therefore, Π can be encoded by a tree T as follows: leaves of T are marked with values in the range of f, and each internal node of T is marked with a name of a player. The run of Π starts at the root of T. Whenever an internal node is reached, player P_i, whose name marks that node, is to take the next step. For each input value in D_i, the protocol Π specifies a probability distribution according to which the next node, a child of the present one, is selected (assuming P_i is honest). The key observation, beyond the technique of [4], is that these distributions (together with the input distribution over D_i) induce a single distribution for the next move of (honest) player i, conditioned on the execution having reached the present node. The outcome of this process is determined by the leaf it reaches (i.e., $f = u$, where u is the mark of the leaf that is reached).

For the analysis, let z be an internal node of T, and consider the same process as above, performed on the subtree of T rooted at z. Suppose that coalition S plays its best strategy to make the outcome $f = v$ most likely, and let $\phi_S^{<z>}$ be that maximum probability (clearly, when z is taken to be the root of T, then $\phi_S^{<z>} = \phi_S$). The key step in the proof is to establish the following inequality for every internal z:

$$\prod_{|R|=t} \phi_R^{<z>} \ge p_{v,z}^{\binom{m-1}{t}}, \tag{1}$$

where $p_{v,z}$ is the probability of reaching a v-marked leaf on that subtree, when all players are honest. Extracting the $\binom{m}{t}$th root of the above inequality, we get $\max_{|R|=t} \phi_R^{<z>} \geq p_{v,z}^{(m-t)/m}$. Taking z to be the root of T the theorem follows.

Inequality (1) is proven by induction on the distance from the leaves in T. In the induction step, we assume that the inequality holds for the children of an internal node z and derive the inequality for node z. Let I denote the set of edges emanating from z and let $\{z_i : i \in I\}$ denote the corresponding children. Suppose, without loss of generality, that node z is marked by player 1. The protocol Π and the probability distributions on the sets D_i determine the probabilities, $\{\lambda_i > 0 : i \in I\}$, governing the player's next move provided that the player is honest and conditioned on the execution having reached node z. (This distribution may not be easy to determine, but we only need to know that it exists.) Now, clearly $p_{v,z} = \sum_{i \in I} \lambda_i p_{v,z_i}$ and $\phi_R^{<z>} = \sum_{i \in I} \lambda_i \phi_R^{<z_i>}$, for every coalition R that does not contain player 1. On the other hand, for every coalition R which does contain player 1, we have $\phi_R^{<z>} = \max_{i \in I} \phi_R^{<z_i>}$. Now, denoting $\phi_R^{<z_i>}$ by $a_{i,R}$ (where $R \subseteq [m]$, $|R| = t$) and p_{v,z_i} by b_i, the inductive step reduces to proving the following numerical lemma, which in turn is a generalization of Lemma 5.3 in [4].

LEMMA 3.3. *Let I be a finite set, let $\{a_{i,R} : i \in I, R \subseteq [m], |R| = t\}$, $\{b_i : i \in I\}$ be nonnegative reals, let $\{\lambda_i : i \in I\}$ be positive with $\sum_{i \in I} \lambda_i = 1$, and assume that for every $i \in I$,*

$$\prod_{R \subseteq [m], |R|=t} a_{i,R} \geq b_i^{\binom{m-1}{t}}.$$

Furthermore, let α_R equal $\max_{i \in I} a_{i,R}$ if $1 \in R$ and $\sum_{i \in I} \lambda_i a_{i,R}$ otherwise. Also, let $\beta = \sum_I \lambda_i b_i$. Then,

$$\prod_{R \subseteq [m], |R|=t} \alpha_R \geq \beta^{\binom{m-1}{t}}.$$

Lemma 5.3 in [4] is a special case of Lemma 3.3 (in which $|I| = 2$ and $\lambda_1 = \lambda_2 = \frac{1}{2}$). However, the ideas presented in the proof of Lemma 5.3 in [4] suffice for proving the general case. In fact, we further generalize Lemma 3.3.

LEMMA 3.4. *Let J, K, and I be disjoint finite sets, let $\{a_{i,j} | i \in I, j \in J \cup K\}$, $\{b_i | i \in I\}$ be nonnegative reals, let $\{\lambda_i | i \in I\}$ be positive, with $\sum_{i \in I} \lambda_i = 1$, and assume that for every $i \in I$,*

$$\prod_{j \in J \cup K} a_{i,j} \geq b_i^{|K|}.$$

For every $j \in J$, let α_j equal $\max_{i \in I} a_{i,j}$ and for every $k \in K$, let $\alpha_k = \sum_{i \in I} \lambda_i a_{i,k}$. Also $\beta = \sum_I \lambda_i b_i$. Then,

$$\prod_{j \in J \cup K} \alpha_j \geq \beta^{|K|}.$$

Lemma 3.3 follows from Lemma 3.4 by letting J be the set of all t-subsets of $[m]$ which contain the element 1 and K be the set of all t-subsets which do not contain 1.

Proof of Lemma 3.4. There is, of course, no loss in assuming

$$b_i = \left(\prod_{j \in J \cup K} a_{i,j} \right)^{1/|K|}$$

for every $i \in I$. Fix all $a_{i,j}$ (over all $i \in I, j \in J$) as well as all $a_{i,k}$ (all $i \in I, k \in K \setminus \{k_1, k_2\}$). Now consider the minimum of $(\sum_{i\in I} \lambda_i a_{i,k_1})(\sum_{i\in I} \lambda_i a_{i,k_2})$ subject to the condition that $a_{i,k_1} \cdot a_{i,k_2}$ are fixed, for all i. A simple calculation with Lagrange multipliers shows that the vectors $(a_{i,k_1}|i \in I)$ and $(a_{i,k_2}|i \in I)$ are proportionate. In other words, there is a nonnegative vector $(u_i|i \in I)$ and nonnegative constants $\rho_k (k \in K)$ such that $a_{i,k} = \rho_k \cdot u_i$ for every $i \in I, k \in K$. Multiply by λ_i and sum over $i \in I$ to conclude that for any $k \in K$, $\alpha_k = \rho_k \sum_I \lambda_i u_i$. We can write now, for every $i \in I$:

$$\left(\prod_{j\in J} \alpha_j\right)^{1/|K|} = \left(\prod_{j\in J} (\max_{i\in I} a_{i,j})\right)^{1/|K|} \geq \left(\prod_{j\in J} a_{i,j}\right)^{1/|K|}$$

and,

$$\left(\prod_{k\in K} \rho_k\right)^{1/|K|} u_i = \left(\prod_{k\in K} a_{i,k}\right)^{1/|K|}.$$

So, for every $i \in I$,

$$\left(\prod_{j\in J} \alpha_j\right)^{1/|K|} \left(\prod_{k\in K} \rho_k\right)^{1/|K|} u_i \geq \left(\prod_{j\in J\cup K} a_{i,j}\right)^{1/|K|} = b_i. \tag{2}$$

Multiply equation (2) by $\rho_t \lambda_i$, sum over $i \in I$, and use $\alpha_t = \rho_t \sum_{i\in I} \lambda_i u_i$ and $\beta = \sum_{i\in I} \lambda_i b_i$, to conclude that for every $t \in K$,

$$\left(\prod_{j\in J} \alpha_j\right)^{1/|K|} \left(\prod_{k\in K} \rho_k\right)^{1/|K|} \alpha_t \geq \rho_t \cdot \beta.$$

Now multiply over all $t \in K$ to get the desired conclusion. □

4. Two-party protocols. In this section we present protocols which meet the lower bounds presented in section 3, up to a polylogarithmic factor. We first present a general framework for the construction of such protocols (subsection 4.1), argue that this framework does indeed yield protocols meeting the lower bound (subsection 4.2), and finally use the framework to present *efficient* protocols meeting the lower bound (subsection 4.3).

Without loss of generality, we assume throughout that every value v in the range of f appears in each row and column in the matrix of f at least $\frac{p_v}{4} \cdot 2^n$ times. If some row or column has too few occurrences of v, we'd like to augment them, without a significant increase in q_v. This can be done as follows: Let $(A_1, \ldots, A_k)$ be a partition of $\{1, \ldots, 2^n\}$, where each A_i has cardinality between $\frac{p_v}{4} \cdot 2^n$ and $\frac{p_v}{2} \cdot 2^n$. It is easy to see that by changing some elements within the $A_i \times A_i$ minors of the matrix to v, it is possible to guarantee that v-values have density $\geq \frac{p_v}{4}$ in every row and column without increasing the largest density in any row or column beyond $q_v + \frac{p_v}{4} = O(q_v)$.

Also, without loss of generality, we assume $p_v \leq 1/2$ (otherwise, the claims hold vacuously).

4.1. Framework for protocols meeting the lower bounds. The goal of the protocol is to enable the parties to gradually reveal their inputs to each other, without granting any party a substantial influence on the value of f.

The protocol proceeds in rounds, each consisting of two steps. In each step one party sends one bit of information about its input to the other party. In the next step the other party sends such a bit. The bits sent by each party specify in which side, of a bipartition of the residual input space, its actual input lies. These partitions must satisfy some "value-balance" properties to be discussed below. Following is the code of the *generic protocol*.

> *Inputs:* $x \in X_0 \stackrel{\text{def}}{=} \{0,1\}^n$ for the row player, $y \in Y_0 \stackrel{\text{def}}{=} \{0,1\}^n$ for the column player.
>
> *Round i:* Let (X^0_{i-1}, X^1_{i-1}) be a partition of X_{i-1}, and (Y^0_{i-1}, Y^1_{i-1}) a partition of Y_{i-1}.
>
> The row player sends $\sigma \in \{0,1\}$ such that $x \in X^\sigma_{i-1}$. Let $X_i \stackrel{\text{def}}{=} X^\sigma_{i-1}$.
>
> The column player sends $\sigma \in \{0,1\}$ such that $y \in Y^\sigma_{i-1}$. Let $Y_i \stackrel{\text{def}}{=} Y^\sigma_{i-1}$.
>
> *Output:* When both residual sets become singletons (i.e., $|X_t| = |Y_t| = 1$ after round t) the protocol terminates and the output is defined as $f(x,y)$, where $X_t = \{x\}$ and $Y_t = \{y\}$).

The reader may think of the partitions as splitting the current set evenly and, in fact, this is almost the case as asserted in Property (P0). In such a case, the protocol terminates after n rounds. For the protocol to achieve its goal (of minimizing the advantage of each party), it employs bipartitions satisfying various (additional) *value-balance* properties. There will be several different types of value-balance properties all sharing the following features, and being applied to both row partitions and column partition. A typical row-partition property (resp., column-partition property) requires that a subset of the rows (resp., columns), specified by some pattern of v-entries, is split almost evenly between the two sides of the partition. For example, Property (P1) below (regarding column-partitions) requires that, for each row, the set of columns containing a v-entry in this row is split almost evenly.

We will introduce the various properties in an ad-hoc manner, each property being introduced just where it becomes essential for analyzing the generic protocol. Thus, at the end of this subsection, we will have a set of properties and a proof that if the protocol utilizes only partitions having these properties, then the advantage of both parties is bounded as claimed in the introduction. The question of whether such partitions exist will be ignored altogether in the current subsection but will be the focus of the next subsection, whereas the third subsection shows how to efficiently generate "pseudorandom" partitions which satisfy these properties.

4.1.1. Motivation to the analysis of the protocol. In analyzing the influence of a dishonest party we consider, without loss of generality, the probability that the row player (following an arbitrary adversarial strategy) succeeds in having the protocol yield a particular value v (in the range of f). For simplicity, we consider first the special case where $q_v = p_v$. In this case there are exactly $K \stackrel{\text{def}}{=} p_v \cdot N$ entries of value v in each row of the matrix. The analysis proceeds in three stages:

> *Stage* 1. Consider the first $\log_2 K$ rounds. If every column (resp., row) partition employed halves the number of v-entries in each row (resp., column), then at the end of this stage the residual $1/p_v$-by-$1/p_v$ matrix contains a single v-entry in each row (resp., column), thus preserving the density of v-entries in each row and column. Using a v-balance property of the partitions called (P1), we show that this is roughly the situation (see Corollary 4.6).

Stage 2. Consider the next $\frac{1}{2} \cdot \log_2(1/p_v)$ rounds. If each row (resp., column) partition employed halves the number of v-entries in the residual matrix, then at the end of this stage the residual $\frac{1}{\sqrt{p_v}}$-by-$\frac{1}{\sqrt{p_v}}$ matrix contains a single v-entry, thus preserving the density of v-entries. Using a v-balance property of the partitions called (P2), we show that this is roughly the situation (see Lemma 4.7).

Stage 3. At the last $\frac{1}{2} \cdot \log_2(1/p_v)$ rounds the row player can force the outcome to be v only if the input of the column player is a column containing a v-entry. The probability that the input column of the column player contains a v-entry does not exceed $\Delta \cdot \sqrt{p_v}$, where Δ is the number of v-entries at the outset of this stage.

4.1.2. Preliminaries. All value-balance properties are geared to guarantee an "almost even split" of certain sets. This is quantified in the following definition with bounds that depend on the size of the set to be split. The size ranges are parameterized by b. For sets smaller than b we require nothing. For sets larger than b^4 we require sublinear discrepancy/bias, and in the midrange we require a small-but-linear discrepancy.

DEFINITION 4.1 (almost unbiased partitions). *Let $S \subseteq U$ be finite sets and $b > 1$. A partition (U^0, U^1) of U is at most b-biased with respect to S if*

(1) *If $|S| \geq b^4$ then $\left||U^0 \cap S| - \frac{|S|}{2}\right| < |S|^{3/4}$.*

(2) *If $b < |S| < b^4$ then $\left||U^0 \cap S| - \frac{|S|}{2}\right| < \frac{|S|}{20}$.*

In our analysis of the protocol, we assume that it utilizes partitions which are at most $\delta \cdot \log_2(1/p_v))$-biased with respect to specific sets, where δ is a constant to be determined as a function of other constants which appear in the analysis (see subsections 4.2 and 4.3). We stress that p_v denotes the density of v-entries in the original matrix corresponding to the function f (and not the density in any residual submatrices defined by the protocol). We denote $\Delta_v \stackrel{\text{def}}{=} \delta \log_2(1/p_v)$. Whenever obvious from the context, we abbreviate Δ_v by Δ.

In addition to value-balance properties, we use the following more elementary property asserting that the partitions are into almost equal sizes. The parameter of approximation is determined by the frequency of the value being discussed in the context.

DEFINITION 4.2 (balance property P0). *A partition (U^0, U^1) of U is said to have Property* (P0) (with respect to a parameter Δ) *if the partition is at most Δ-biased with respect to U. When $|U| \geq 2$ it is also required that the partition be nontrivial; namely $|U^0|, |U^1| \geq 1$.*

The additional condition guarantees that if the generic protocol uses only partitions with Property (P0) then it terminates. The main condition in Property (P0) implies termination in at most $n+\Delta$ rounds (see Claim 4.4 and the proof of Lemma 4.5).

We consider executions of the generic protocol under various strategies of the row player, typically assuming that the column player plays honestly. The *residual submatrix* after i rounds is the submatrix corresponding to $X_i \times Y_i$. We denote by $\#_v(X, Y)$ the number of v-entries in the submatrix induced by $X \times Y$. When X is a singleton, $X = \{x\}$, we abbreviate and write $\#_v(x, Y)$ instead of $\#_v(X, Y)$. For example, for $x \in X_i$, the number of v-entries in the residual x-row after i rounds (resulting in the residual submatrix $X_i \times Y_i$) is denoted $\#_v(x, Y_i)$.

4.1.3. Analysis of the protocol: The special case of $q_v = p_v$. For the analysis of this special case, we need two types of "value-balance" properties. The

definition is phrased for column partition. An analogous definition holds for row partitions.

DEFINITION 4.3 (value-balance properties P1 and P2). *Let X_i and Y_i be residual sets of rows and columns and let (Y_i^0, Y_i^1) be a* (column) *partition of Y_i, and v be a value in the range of f. We consider the following two properties:*

Property (P1). *The partition is* v-balanced with respect to individual rows *if the following holds. For every* (remaining) *row $x \in X_i$, the partition is at most Δ_v-biased with respect to set of columns having v-entries in row x* (i.e., w.r.t. the sets $\{y \in Y_i : f(x,y) = v\}$, for each $x \in X_i$).

Property (P2). *Either $|Y_i| \geq 2/p_v$ or the partition is* v-balanced with respect to the standard coloring *in the following sense. Consider a standard* minimum *coloring, ξ, of the v-entries in $X_i \times Y_i$, where no two v-entries in the same column or row are assigned the same color. For every color α, the partition is at most Δ_v-biased with respect to the set of columns containing a v-entry of color α* (i.e., w.r.t. the sets $\{y \in Y_i : \exists x \in X_i \text{ s.t. } f(x,y) = v \text{ and } \xi(x,y) = \alpha\}$, over $\alpha \in \text{Range}(\xi)$).

The following is an elementary technical claim, which we use extensively in the analysis.

CLAIM 4.4. *Let $\alpha < 1$. Suppose that $z_{i+1} < \frac{z_i}{2} + (z_i)^\alpha$, for every $i = 0, \ldots, T$. Then, there exists a constant c_α, so that $z_t < \frac{z_0}{2^{t-1}}$, for every $t < \min\{T, (\log_2 z_0) - c_\alpha\}$. Likewise, if $z_{i+1} > \frac{z_i}{2} - (z_i)^\alpha$, for every $0 \leq i \leq T$, then $z_t > \frac{z_0}{2^{t+1}}$, for every $t < \min\{T, (\log_2 z_0) - c_\alpha\}$.*

Proof. By successively applying the inequality t times, we get $z_t < \frac{z_0}{2^t} + \sum_{i=1}^{t} \frac{z_{t-i}^\alpha}{2^{i-1}}$. Using induction on t, we get

$$\begin{aligned} z_t &< \frac{z_0}{2^t} + \sum_{i=1}^{t} \frac{(z_0/2^{t-i-1})^\alpha}{2^{i-1}} \\ &= \frac{z_0}{2^t} + 2 \cdot \left(\frac{2z_0}{2^t}\right)^\alpha \cdot \sum_{i=1}^{t} \left(\frac{1}{2^{1-\alpha}}\right)^i \\ &< \frac{z_0}{2^t} + 2^{1+\alpha} \cdot \left(\frac{z_0}{2^t}\right)^\alpha \cdot \frac{1}{2^{1-\alpha} - 1}, \end{aligned}$$

which is bounded by $\frac{z_0}{2^{t-1}}$, provided that $\frac{z_0}{2^t} > 2^{c_\alpha}$ where $c_\alpha \stackrel{\text{def}}{=} \frac{1}{1-\alpha} \cdot \log_2(2^{1+\alpha}/(2^{1-\alpha} - 1))$. □

We start by showing that the density of v-entries in individual rows and columns hardly changes as long as each such row/column contains enough v-entries and the partitions split them almost evenly. This assertion corresponds to stage (1) in the motivating discussion.

LEMMA 4.5 (stage 1). *Let v be a value in the range of f, and suppose that the protocol uses* column *partitions satisfying Property* (P1) w.r.t. the value v. *Let K_x denote the number of v entries in the original row x. Then, regardless of the players' steps, if row x is in the residual matrix after the first $i \stackrel{\text{def}}{=} \log_2 K_x$ rounds, then there are at most Δ_v residual v-entries in row x.* (i.e., $\#_v(x, Y_i) \leq \Delta_v$). *Furthermore, after $t < K_x$ rounds $\#_v(x, Y_t) \leq \Delta_v \cdot 2^{K_x - t}$.*

Proof. The analysis uses the fact that the column partitions are v-balanced with respect to each row. Using condition (1) of the almost unbiased property (Def. 4.1) and Claim 4.4, we see that after the first $s \stackrel{\text{def}}{=} \log_2 K_x - 4 \log_2 \Delta$ rounds the residual row x has at most $\frac{K_x}{2^{s-1}} = 2\Delta^4$ entries of value v. For the remaining $r \stackrel{\text{def}}{=} 4 \log_2 \Delta$ rounds we use condition (2) of the almost unbiased property, to show that the number

of v-entries in the row is at most Δ. This follows by considering r iterations of condition (2), namely,

$$\begin{aligned} 2\Delta^4 \cdot \left(\frac{1}{2}+\frac{1}{20}\right)^{4\log_2\Delta} &= 2\cdot\left(1+\frac{1}{10}\right)^{4\log_2\Delta} \\ &= 2\cdot\Delta^{4\log_2(1+\frac{1}{10})} \\ &< 2\cdot\Delta^{2/3} \\ &\le \Delta, \end{aligned}$$

where in the last inequality we use $\delta \ge 8$ (and $p_v \le 1/2$). The lemma follows. □

As an immediate corollary, we get the following.

COROLLARY 4.6 (stage 1 for $q_v = p_v$). *Let $v \in Range(f)$, and suppose that $q_v = p_v$. Suppose that the protocol uses* column (resp., row) *partitions satisfying Property* (P1) w.r.t. the value v. *Then after the first $n - \log_2(1/p_v)$ rounds, the number of v-entries in each residual row* (resp., column) *is at most Δ_v $(= \delta \cdot \log 1/p_v)$. This statement holds regardless of the steps taken by the players.*

Proof. Observe that $q_v = p_v$ implies that each (original) row has $p_v \cdot 2^n$ entries of value v, and apply Lemma 4.5. □

When the number of v-entries in individual rows and columns is small, but not too small, we'd like to assert something in the spirit of stage (2) of the motivating discussion. Namely, that the density of v-entries in the *entire* matrix is preserved as long as their total number is not too small and the partitions behave nicely w.r.t the existing v-entries.

LEMMA 4.7 (stage 2). *Let $M < 2/p_v$. Consider an M-by-M matrix where no row or column contains more than B v-entries. Suppose that the protocol is applied to this matrix, using* column and row *partitions that satisfy Property* (P2) w.r.t. the value v. *Then, after the first $\frac{1}{2}\log_2 M$ rounds, the number of v-entries in the residual submatrix is at most $(2B+1)\cdot\Delta_v$. This statement holds regardless of the steps taken by the players.*

Proof. The analysis uses only the fact that the row and column partitions are v-balanced with respect to the standard coloring. (The upper bound on M implies that this is the only way to satisfy Property (P2).) Note that the standard coloring, being a minimum coloring, uses at most $2B+1$ colors since the underlying graph has maximum degree $\le 2B$. Let α be a color. In each row and column there is at most one v-entry of color α, hence each row/column partition approximately halves the number of remaining v-entries of color α. Hence, using the same arguments as in Lemma 4.5, we see that after $\frac{1}{2}\log_2 M$ rounds the residual matrix contains at most Δ_v v-entries of color α. The lemma follows. □

Finally, when the total number of v-entries in the residual matrix is small we observe that v may be the output only if the input of the column player corresponds to a residual column containing a v-entry. This corresponds to stage (3) in the motivating discussion. Thus, using Corollary 4.6 and Lemma 4.7, we get the following.

COROLLARY 4.8 (advantage in case $q_v = p_v$). *Let $q_v = p_v$ for $v \in Range(f)$. Suppose that the protocol uses only partitions that satisfy Properties* (P0), (P1) *and* (P2) *w.r.t. v. Then the protocol outputs v with probability at most $O(\Delta_v^2\sqrt{p_v})$ $(= O((\delta \log 1/p_v)^2\sqrt{p_v}))$, regardless of the row player's steps.*

Proof. Corollary 4.6 and Lemma 4.7 imply that after the first $\log_2(p_v N) + \frac{1}{2}\log_2(1/p_v)$ rounds, the number of v-entries in the residual matrix is at most $O(\Delta^2)$. If in all partitions the two parts have equal size, then the residual matrix has dimen-

sion $\sqrt{1/p_v}$-by-$\sqrt{1/p_v}$. Property (P0) is applied to show that the residual submatrix has size at least $\frac{1}{2}\sqrt{1/p_v}$-by-$\frac{1}{2}\sqrt{1/p_v}$. To this end we use Claim 4.4 and the observation that $\sqrt{1/p_v} > \Delta_v^4 = (\delta \log_2(1/p_v))^4$, provided that p_v is bounded above by some constant. Such a bound on p_v may be assumed, possibly increasing some constants in the O-terms. Finally, we observe that the output of the protocol is v only if the input of the column player specifies a column containing a v-entry in the residual submatrix. The corollary follows. □

Using "sufficiently random" partitions, the above bound can be improved to $O(\sqrt{p_v})$. For details see Theorem 4.27.

4.1.4. Analysis of the protocol: The general case — row classes. The analysis of the general case (where q_v may exceed p_v) is more cumbersome. To facilitate the understanding we precede each technical step by a motivating discussion. As before, we analyze the advantage of the row player towards some value v. Throughout the analysis we introduce additional value-balance properties that the partitions used in the protocol should satisfy for the analysis to proceed. Later in the paper we discuss how to find such partitions and show that "slightly random" partitions do have these properties.

We classify the rows by density and apply the analysis separately to each class. Let $\rho_v(x)$ denote the density of v-entries in row x of the original matrix; that is,

$$\rho_v(x) \stackrel{\text{def}}{=} \frac{|\{y \in Y_0 : f(x,y) = v\}|}{|Y_0|} = \frac{\#_v(x, Y_0)}{|Y_0|}. \tag{3}$$

By our assumption, $\frac{p_v}{4} < \rho_v(x) \leq q_v$, for every $x \in X_0$, and the average of ρ_v, over all $x \in X_0$, equals p_v. For $0 \leq j \leq \log_2(1/p_v) + 1$, define R^j as the class of all rows with v-entry density between 2^{-j} and 2^{-j-1}; that is,

$$R^j \stackrel{\text{def}}{=} \{x \in X_0 : \lfloor \log_2(1/\rho_v(x)) \rfloor = j\}. \tag{4}$$

Note that the last class, $R^{\log_2(1/p_v)+1}$, contains all rows with v-entry density smaller than $p_v/2$.

Clearly, the influence of the row player towards value v is bounded by the sum of its influences (towards v) when restricting itself to inputs/rows of a certain class. Recall that the row player behavior is always restricted (by our hypothesis that it is not detected cheating) to sending a single bit in each round. The assumption that the row player restricts itself to inputs/rows in a particular set means that its answers must be consistent with some input in the set (i.e., in round i he may send σ only if X_i^σ intersects the restricted set). The above is summarized and generalized in the following claim.

CLAIM 4.9. *For $Z \subseteq X$ a set of rows, we let θ_Z be the probability for an outcome of v, assuming that the actions of the row player are consistent with some row in Z, but is otherwise free to choose any adversarial strategy. If $(Z_1, \ldots, Z_r)$ is a partition of the set of rows, then the probability for the protocol to have outcome v does not exceed $\sum_i \theta_{Z_i}$.*

Proof. The claim follows applying a union bound. □

We now partition the row classes into two categories: *heavy* rows with density above $\sqrt{p_v}$ and rows below this density. First, we bound the advantage of the row player when it restricts itself to heavy inputs/rows. A simple counting argument implies that there are at most $\sqrt{p_v}N$ heavy rows. We will consider the situation after $\log_2(\sqrt{p_v}N)$ rounds of the protocol. Using an additional v-balance property, denoted

P3, which asserts that the row partitions split almost evenly the set of heavy rows, we will show that after $\log_2(\sqrt{p_v}N)$ rounds at most Δ of the heavy rows remain in the residual matrix and furthermore that each such row maintains its original v-density up to a multiplicative factor of Δ. Loosely speaking, the row player can now choose only between Δ possible inputs/rows with probabilities of success that equal the density of the residual row. Thus, the advantage of the row player (towards v) when restricting itself to heavy rows is bounded by $\Delta^2 \cdot q_v = O((\log_2(1/p_v))^2 q_v)$.

DEFINITION 4.10 (value-balance property P3). *Let X_i and Y_i be residual sets of rows and columns and and let $v \in Range(f)$. A row partition has Property* (P3) (is said to be v-balanced with respect to heavy rows) *if it is at most Δ_v-biased with respect to the set of the* (remaining) *heavy rows* (i.e., w.r.t. the set $\{x \in X_i : \rho_v(x) \geq \sqrt{p_v}\}$).

LEMMA 4.11 (advantage via heavy-row strategies). *Suppose that the protocol is performed using* column and row *partitions satisfying Properties* (P0), (P1), and (P3) w.r.t. the value v. *Then, as long as the row player restricts itself to heavy rows and the column player plays honestly, the output equals v with probability at most $2\Delta_v^2 \cdot q_v$.*

Proof. Consider the situation after $\log_2(\sqrt{p_v}N)$ rounds of the protocol. Heavy rows have at least $\sqrt{p_v}N$ entries of value v and so we will be able to apply Lemma 4.5 to these rows. Using Property (P1) and applying Lemma 4.5 to each heavy row, we conclude that every remaining heavy row x contains at most $\Delta \cdot 2^i$ v-entries, where

$$\begin{aligned} i &\stackrel{\text{def}}{=} \log_2(\rho_v(x)N) - \log_2(\sqrt{p_v}N) \\ &\leq \log_2(q_v N) - \log_2(\sqrt{p_v}N) \\ &= \log_2(q_v/\sqrt{p_v}). \end{aligned}$$

(We are assuming that heavy rows exist, i.e., $q_v \geq \sqrt{p_v}$, whence $i \geq 0$.) Thus, each such heavy row contains at most $\Delta \cdot q_v/\sqrt{p_v}$ v-entries. Also, using Property (P3) and an argument as in the proof of Lemma 4.5, it follows that the residual matrix has at most Δ heavy rows. Thus, the entire residual matrix contains at most $\Delta^2 \cdot q_v/\sqrt{p_v}$ *v-entries in heavy rows*. Using Property (P0) we know that the residual matrix at this stage contains at least $\frac{1}{2}\sqrt{1/p_v}$ columns. Thus, by an argument as in the proof of Corollary 4.8, the probability that the protocol terminates with a pair (x, y) so that x is heavy and $f(x, y) = v$ does not exceed

$$\frac{\#_v(H \cap X_i, Y_i)}{|Y_i|} \leq \frac{\Delta^2 \cdot q_v/\sqrt{p_v}}{1/(2\sqrt{p_v})} = 2\Delta^2 q_v,$$

where H is the set of heavy rows and $X_i \times Y_i$ is the residual matrix. The lemma follows. □

Having analyzed strategies where the row player confines itself to heavy rows, we turn to strategies where it refrains from heavy rows. The analysis is split according to the remaining row classes; that is, for every $1 \leq j \leq \frac{1}{2}\log_2(1/p_v)$, we bound the advantage of the row player assuming that it restricts itself to the class (of rows) $R \stackrel{\text{def}}{=} R^{j+\frac{1}{2}\log_2(1/p_v)}$ that have density $\approx \sqrt{p_v}2^{-j}$. By a counting argument,

$$|R| \leq \sqrt{p_v}2^j N. \tag{5}$$

Consider the situation after $\log_2(\sqrt{p_v}2^{-j}N)$ rounds. Note that this corresponds to stage (1) in the motivating discussion and thus we can apply Lemma 4.5 and assert that after these $\log_2(\sqrt{p_v}2^{-j}N)$ rounds no residual row of R has more than Δ v-entries. Using an additional v-balance property, denoted **P4**, which asserts that the

row partitions split R almost evenly, we will show that after these $\log_2(\sqrt{p_v}2^{-j}N)$ rounds the residual matrix contains at most $\max\{\Delta, 2^{2j+1}\}$ rows of R.

DEFINITION 4.12 (value-balance property P4). *Let X_i and Y_i be residual sets of rows and columns and $v \in Range(f)$. A row partition has Property* (P4) (is said to be v-balanced with respect to row-density classes) *if, for every j ($\frac{1}{2}\log_2(1/p_v) \leq j \leq 1 + \log_2(1/p_v)$), it is at most Δ_v-biased with respect to the set of the* (remaining) *rows in R^j* (i.e., w.r.t. the sets $\{x \in X_i : \lfloor \log_2 \rho_v(x) \rfloor = j\}$, for $\frac{1}{2}\log_2(1/p_v) \leq j \leq 1 + \log_2(1/p_v)$).

LEMMA 4.13 (strategies restricted to $R = R^{j+\frac{1}{2}\log_2(1/p_v)}$ — the first rounds). *Let $v \in Range(f)$, and suppose that the protocol uses* row and column *partitions satisfying Properties* (P0), (P1), *and* (P4) *w.r.t. the value v. Then after the first $i \stackrel{\text{def}}{=} n - j - \frac{1}{2}\log_2(1/p_v)$ rounds, the resulting $X_i \times Y_i$ submatrix satisfies the following conditions, regardless of the players' steps:*

1. *each remaining row of R contains at most Δ_v entries of value v (i.e., $\#_v(x, Y_i) \leq \Delta_v$, for every $x \in R \cap X_i$);*
2. *at most $\Delta_v \cdot 2^{2j+1}$ rows of R remain (i.e., $|R \cap X_i| \leq \Delta_v \cdot 2^{2j+1}$);*
3. *the number of columns is at least $\frac{1}{2} \cdot \frac{2^j}{\sqrt{p_v}}$ (i.e., $|Y_i| \geq \frac{1}{2} \cdot \frac{2^j}{\sqrt{p_v}}$).*

Proof. Item (1) follows from Lemma 4.5 (using Property (P1)). Using Property (P4), we derive item (2) as in the second part of the proof of Lemma 4.11. Finally, item (3) follows using Property (P0). □

For "small" j's (say, $j \leq \log_2 \Delta$) we get into a situation as in the analysis of heavy rows. Actually, the following applies to any j, but is useful only for $j = O(\log \Delta_v)$.

COROLLARY 4.14 (advantage via $R = R^{j+\frac{1}{2}\log_2(1/p_v)}$ strategies — simple analysis). *Consider a protocol in which all* column and row *partitions satisfy Properties* (P0), (P1), and (P4) w.r.t. the value v. *Then, as long as the row player restricts itself to rows in R and the column player plays honestly, the output equals v with probability at most $2^{j+2}\Delta_v^2 \cdot \sqrt{p_v}$.*

Proof. Using Lemma 4.13 we infer that the residual matrix after i rounds has at most $\Delta^2 \cdot 2^{2j+1}$ v-entries in rows of R and at least $\frac{1}{2} \cdot \frac{2^j}{\sqrt{p_v}}$ columns. Thus, the probability that the column chosen by the column player has a v-entry in a residual row of R does not exceed

$$\frac{\#_v(R \cap X_i, Y_i)}{|Y_i|} \leq \frac{\Delta^2 \cdot 2^{2j+1}}{\frac{1}{2} \cdot \frac{2^j}{\sqrt{p_v}}} = 2^{j+2}\Delta^2\sqrt{p_v}.$$

The corollary follows. □

So far we dealt with heavy rows and the row classes $R^{j+\frac{1}{2}\log_2(1/p_v)}$ for "small" j's, $j \leq \log_2 \Delta_v$. The rest of the analysis concentrates on row classes $R^{j+\frac{1}{2}\log_2(1/p_v)}$ for $j > \log_2 \Delta_v$.

4.1.5. Analysis of the protocol: The general case — column subclasses. Lemmas 4.11 and 4.13 summarize what we can infer by considering only row classes defined by the density of v-entries. We learned that after $i = n - j - \frac{1}{2}\log_2(1/p_v)$ rounds the resulting matrix has approximately 2^{2j} rows of the class $R = R^{j+\frac{1}{2}\log_2(1/p_v)}$ with no more than Δ v-entries in each such row. Thus, in total the resulting submatrix has approximately 2^{2j} v-entries in rows of R. Had these v-values been distributed evenly among the columns, then we could apply an argument analogous to Lemma 4.7 (corresponding to stage (2) in the motivating discussion). At the other extreme, if these v-values are all in one column, then we should have further applied Lemma 4.5

to this column. In general, however, the distribution of these v-entries may be more complex and in order to proceed we classify columns according to the approximate density of v-entries within each particular row class. Once this is done, the matrix is split to submatrices such that the density of v-entries in each induced subcolumn is about the same. Each such submatrix is easy to analyze and we can combine these analyses to derive the final result.

Let $\ell \stackrel{\text{def}}{=} \frac{1}{2}\log_2(1/p_v)$. Recall that we are currently dealing with an arbitrary $R = R^{j+\ell}$, where $1 < j \leq \ell + 1$. For $0 \leq k \leq 2j$, let

$$C_j^k \stackrel{\text{def}}{=} \{y \in Y_0 : \lfloor \log_2(1/\mu_v(y, R^{j+\ell})) \rfloor = k\}, \tag{6}$$

where $\mu_v(y, R)$ is the density of v-entries in the *portion* of column y restricted to rows R, that is

$$\mu_v(y, R) = \frac{|\{x \in R : f(x, y) = v\}|}{|R|}. \tag{7}$$

Columns having lower v-density within R (i.e., $\mu_v < 2^{-2j-1}$) are defined to be in $C_j = C_j^{2j+1}$ and will be treated separately. The advantage of the row player towards v when restricting its input to R is the sum, over all k, of the probabilities of the following $2j + 2$ events. For $k = 0, \ldots, 2j + 1$, the kth *event* occurs if the input of the column player happens to be in C_j^k *and* the output of the protocol is v (when the row player restricts its input to be in R). Thus, it suffices to bound the probability of each of these $2j + 2$ events. We first observe that, for $j = 0, \ldots, l + 1$ and $0 \leq k \leq 2j$,

$$|C_j^k| \leq \frac{\#_v(R^{j+\ell}, Y_0)}{\min_{y \in C_j^k}\{\#_v(R^{j+\ell}, y)\}} \leq \frac{|R^{j+\ell}| \cdot (\sqrt{p_v} 2^{-j} \cdot N)}{2^{-k-1} \cdot |R^{j+\ell}|} = 2^{k+1-j}\sqrt{p_v} \cdot N. \tag{8}$$

Thus, the probability that the input of the column player is in C_j^k is bounded by $2^{k+1-j}\sqrt{p_v}$. This by itself provides a sufficiently good bound for the case $k \leq j$ and so it is left to consider the case where $j < k \leq 2j$ and to deal with the columns in C_j. We start with the latter. (Warning: the next two paragraphs consist of an imprecise motivating discussion; a rigorous treatment follows.)

Considering the submatrix $R \times C_j$ and using item (2) of Lemma 4.13 we know that, after $i = n - j - \ell$ rounds, each residual row in this submatrix contains at most Δ v-entries. Assuming that the row partitions split the v-entries in the subcolumn of this submatrix almost evenly (as postulated in an additional value-balance property, denoted **P6**), we conclude that residual subcolumns of the submatrix contain at most Δ v-entries (note that there are at most 2^{2j+1} rows of R and that the v-density of columns in $R \times C_j$ is at most 2^{-2j-1}). Thus, we can apply an analysis analogous to stage (2) in the motivating discussion. It follows that after an additional j rounds, the resulting submatrix contains at most Δ^2 v-entries. At this stage, there are still $\ell = \frac{1}{2}\log_2(1/p_v)$ rounds to go so we conclude that the probability that the column player's input is in C_j and the output is v (when the row player restricts its input to be in R) is at most $\Delta^2\sqrt{p_v}$. This argument will be made precise as a special case of the argument for C_j^k, $k > j$.

We now consider the submatrix $R \times C$, where $C \stackrel{\text{def}}{=} C_j^k$ for $k > j$. Again, by Property (P6) we expect each residual subcolumn to contain $2^{-k} \cdot 2^{2j}$ entries of value v. Assuming that the column partitions split C almost evenly, as postulated in yet another value-balance property (**P5** below), and using equation (8), we expect the residual submatrix to contain at most 2^{k+1} columns of C_j^k (and, recall, 2^{2j} rows of R).

Thus, the next $2j-k<k$ rounds are expected to preserve the density of C columns in the residual matrix as well as the density of v-entries in residual subcolumns of the submatrix $R\times C$, provided that Properties (P5) and (P6) hold. Thus, at this point (after a total of $(n-j-\ell)+(2j-k)$ rounds) each remaining row of R is left with at most Δ entries of value v and each remaining column of C has at most Δ entries of value v in the portion of the rows of R. Furthermore, we expect the residual $R\times C$ to have $2^{2j-(2j-k)}=2^k$ rows and $2^{k-(2j-k)}=2^{2k-2j}$ columns. We can now apply an argument analogous to Lemma 4.7 (corresponding to stage (2) in the motivating discussion). To this end we introduce the last value-balance property, denoted **P7**, which analogously to (P2) asserts that, with respect to each color in a standard minimum coloring of the v-entries in $R\times C$, the row (resp., column) partitions split almost evenly the set of rows (resp., columns) having v-entries colored by this color. Finally, consider the situation after another additional $k-j$ rounds. Using (P7) in an argument analogous to Lemma 4.7, we show that after these $k-j$ rounds, the residual $R\times C$ submatrix has at most Δ^2 v-entries. Furthermore, this residual submatrix is expected to have $2^{k-(k-j)}=2^j$ rows and $2^{(2k-2j)-(k-j)}=2^{k-j}$ columns. Thus, assuming that the column player's input, denoted y, is in C the probability that it falls in one of the residual columns which has a v-entry in the R-portion is at most $\Delta^2/2^{k-j}$. It follows that the probability for the input column to be in C_j^k *and* the output be v (when the row player restricts its input to R) is at most

$$\frac{\Delta^2}{2^{k-j}}\cdot 2^{k-j}\sqrt{p_v}=\Delta^2\cdot\sqrt{p_v}.$$

Thus, the claimed bound follows also in this case.

We now turn to a rigorous analysis of the advantage of the row player in executions where it restricts itself to inputs in $R=R^{j+\ell}$ and the input column happens to fall in $C\stackrel{\text{def}}{=}C_j^k$, for some $k>j>0$. (Recall that for $k\le j$, equation (8) by itself asserts that input column falls in C_j^k with probability at most $\sqrt{p_v}$.)

DEFINITION 4.15 (value-balance properties P5, P6, and P7). *Let X_i and Y_i be residual sets of rows and columns. Let (X_i^0,X_i^1) be a* row *partition, (Y_i^0,Y_i^1) be a* column *partition, and $v\in Range(f)$. We consider the following three properties.*

Property (P5). *The column partition (Y_i^0,Y_i^1) is* v-balanced with respect to column subclasses *if, for every j,k satisfying $0<j<k\le 2j\le 2\ell+2$, the partition is at most Δ_v-biased with respect to the set of columns in C_j^k* (i.e., w.r.t. the sets $Y_i\cap C_j^k$, for each j,k s.t. $0<j<k\le 2j\le 2\ell+2$).

Property (P6). *For every j and every $y\in Y_i$, either $\frac{\#_v(X_i\cap R^{j+\ell},y)}{|Y_i|}\le\frac{p_v}{4\Delta_v}$ or the row partition (X_i^0,X_i^1) is* v-balanced with respect to the jth subcolumn y *in the sense that the partition is at most Δ_v-biased with respect to the set of rows in $R^{j+\ell}$ having v-entries in y* (i.e., w.r.t. $\{x\in X_i\cap R^{j+\ell}: f(x,y)=v\}$, for each $y\in Y_i$ and j s.t. $0<j\le\ell+1$).

Property (P7). *Either $|Y_i|\ge 4/p_v$ or the partition (Y_i^0,Y_i^1) is* v-balanced with respect to the standard coloring of subclasses *in the following sense. For every j,k as in* (P5)*, consider a standard* minimum *coloring ξ, of the v-entries in $(X_i\cap R^{j+\ell})\times(Y_i\cap C_j^k)$ so that every two v-entries in the same column or row are colored differently. For every color α, the partition is at most Δ_v-biased with respect to the set of columns containing a v-entry of color α* (i.e., w.r.t. the sets $\{y\in Y_i\cap C_j^k:\exists x\in X_i\cap R^{j+\ell}\text{ s.t. }f(x,y)=v\text{ and }\xi(x,y)=\alpha\}$, for each j,k, and α.)

DEFINITION 4.16 (the (j,k)-event). *Let $0 < j \leq \ell + 1$ and $0 \leq k \leq 2j+1$. Fix an arbitrary strategy in which the row player restricts its input to rows in $R^{j+\ell}$. The (j,k)-event* (or kth event) *is said to occur if both the input column is in C_j^k and the output is v.*

LEMMA 4.17 (bounding individual events). *Let $0 < j \leq \ell+1$ and $0 \leq k \leq 2j+1$. Suppose that the protocol uses partitions which satisfy Properties* (P0), (P1), (P4), (P5), (P6), and (P7). *Then, for any strategy in which the row player restricts its input to rows in $R^{j+\ell}$, the probability of the (j,k)-event is at most $5\Delta_v^4 \cdot \sqrt{p_v}$.*

We remark that a lower power of Δ_v can be obtained by a more careful analysis.

Proof. As observed above, the bound holds in case $k \leq j$, since in this case equation (8) implies that the column player's input is in C_j^k with probability at most $\sqrt{p_v}$. We thus turn to the case $j < k \leq 2j+1$.

First, we consider the situation after $i \stackrel{\text{def}}{=} (n-j-\ell) + (2j-k) = n+j-k-\ell$ rounds. Note that $j < k \leq 2j+1$ implies $i \geq (n-j-\ell) - 1 \geq n - \log_2(2/p_v)$ and $i < n - \ell$. We first bound the number of v-entries in the residual subrows and subcolumns of $R \times C$.

Claim 4.17.1. Each remaining row of $R \stackrel{\text{def}}{=} R^{j+\ell}$ contains at most Δ v-entries; namely, $\#_v(x, Y_i) \leq \Delta$, for every $x \in R \cap X_i$.

Proof. Since $i \geq (n-j-\ell)-1$, we can apply Lemma 4.13, and the claim follows by item (1). □

Claim 4.17.2. Each remaining column of $C \stackrel{\text{def}}{=} C_j^k$ contains at most Δ entries of value v within its R-portion; namely, $\#_v(R \cap X_i, y) \leq \Delta$, for every $y \in C \cap Y_i$.

Proof. We first bound the number of v-entries in the R-portion of each column $y \in C$. By combining the definition of C and equation (5), we get

$$\begin{aligned} \#_v(R, y) &\leq 2^{-k} \cdot |R| \\ &\leq \sqrt{p_v} \cdot 2^{j+n-k} \\ &= 2^{j+n-k-\ell} \\ &= 2^i. \end{aligned}$$

We now wish to apply Property (P6) and argue that $\#_v(R \cap X_i, y) \leq \Delta \cdot \#_v(R, y) \cdot 2^{-i}$, but we need to be careful since Property (P6) is useful only when $\#_v(R \cap X_t, y) \geq \frac{p_v}{4\Delta} \cdot |Y_t|$. Thus, before applying Property (P6), we consider the simple case in which there are many v-entries in the R-portion of y; namely, $\#_v(R, y) \geq p_v \cdot |Y_0|$. Using Properties (P6) and (P0), we infer inductively that the ratio $\#_v(R \cap X_i, y)/|Y_i|$ is maintained after $r < i$ rounds. In the induction step we assume that the ratio after r rounds is at least $p_v/2$ and applying Proposition (P6) infer the same for $r+1$ rounds, provided $\#_v(R \cap X_r, y) \geq \Delta^4$. In the last ($\approx 4\log_2 \Delta$) rounds we maintain as invariant the assumption that the ratio is at least p_v/Δ_v. We conclude (analogously to Lemma 4.5) that $\#_v(R \cap X_i, y) \leq \Delta \cdot 2^{i-i} = \Delta$ as claimed. Yet, all the above is valid only in case the initial number of v-entries in the subcolumn is large enough (i.e., $\#_v(R, y) \geq p_v \cdot |Y_0|$), which need not be the case in general. Intuitively, this cannot be a problem since fewer v-entries in the subcolumn can only help. Formally, we proceed as follows. Let $y_0 \stackrel{\text{def}}{=} |Y_0|$ and $z_0 \stackrel{\text{def}}{=} \#_v(R, y)$. Consider i iterations of the following rule:

- If $y_t > \Delta^4$ then set y_{t+1} to be in the interval $[(y_t/2) \pm y_t^{3/4}]$. If $y_t > \Delta$ then set y_{t+1} to be in the interval $[(y_t/2) \pm (y_t/20)]$. Otherwise, set y_{t+1} to be in the interval $[0, y_t]$.

- If $z_t > (p_v/\Delta)\cdot y_t$ then set z_{t+1} analogously to the way y_{t+1} is set. Otherwise, (i.e., $z_t \leq (p_v/\Delta)\cdot y_t$), set z_{t+1} to be in the interval $[0, z_t]$.

The above process corresponds to the decline (with $t = 0, \ldots, i$) of $|Y_t|$ (represented by y_t) and $\#_v(R \cap X_t, y)$ (represented by z_t), as governed by Properties (P0) and (P6). In case the initial ratio z_0/y_0 is sufficiently large, say at least p_v/Δ, Claim 4.4 implies that $z_i \leq \Delta$. As far as the y_t's are concerned, Claim 4.4 can be applied to yield $y_i \leq \Delta \cdot 2^{\ell+k-j}$, which by $k \leq 2j+1$ and $j \leq \ell+1$ yields $y_i \leq 2\Delta \cdot (1/p_v)$. Thus, it is clear that z_i is bounded by the maximum of the bound obtained in the simple case (i.e., Δ) and $(p_v/\Delta)\cdot y_i \leq 2$. The claim follows. □

We are now in a situation analogous to the end of stage (1) in the motivating discussion, except that the bounds on v-entries hold with respect to the residual $R \times C$ submatrix (rather than to the entire residual matrix). Our goal is to now apply a process analogous to stage (2) in the motivating discussion. To this end we first consider a minimum coloring of the v-entries in this residual submatrix (i.e., a coloring in which no v-entries in the same row/column are assigned the same color). Using Claims 4.17.1 and 4.17.2, we first observe that this coloring requires at most $2\Delta + 1$ colors (since the degrees in the induced graph do not exceed 2Δ). Next we derive an upper bound on the size of independent sets in the graph, (i.e., on individual color classes in this coloring). An independent set in this graph meets every row and column at most once, so its cardinality cannot exceed $\min\{|R \cap X_i|, |C \cap Y_i|\}$.

Claim 4.17.3. $\min\{|R \cap X_i|, |C \cap Y_i|\} \leq 2\Delta \cdot 2^{2(k-j)}$.

Proof. Using Property (P4) and equation (5), we get $|R \cap X_i| \leq \Delta \cdot 2^{(n+j-\ell)-i} = \Delta \cdot 2^k$, so the claim holds when $k \geq 2j-1$ and in particular for the class $C_j = C_j^{2j+1}$. Likewise, using Property (P5) and equation (8) and assuming $k \leq 2j$, we get $|C_j^k \cap Y_i| \leq \Delta \cdot 2^{(n+k+1-j-\ell)-i} = \Delta \cdot 2^{2(k-j)+1}$. This proves the claim for the range $k \leq 2j$. □

We now consider an execution of the next $(k-j)$ rounds. Using Property (P7), we proceed analogously to Lemma 4.7. First, we upper bound the size of each residual color class by $\Delta \cdot \frac{2\Delta 2^{2(k-j)}}{2^{2(k-j)}} = 2\Delta^2$ (essentially, its size after i rounds divided by a factor of 2 for each of the $2(k-j)$ steps in the next $k-j$ rounds). Adding up the bounds for all color classes, we obtain a bound on the total number of v-entries in the resulting $R \times C$ submatrix; namely,

$$\#_v(X_{i+k-j} \cap R, Y_{i+k-j} \cap C) \leq (2\Delta+1)\cdot 2\Delta^2 < 5\Delta^3. \tag{9}$$

We are now in a situation analogous to the end of stage (2) in the motivating discussion. We note that till now $i + (k-j) = n - \ell$ rounds were performed. We distinguish two cases.

Case 1. If $|C| < \Delta^3\sqrt{p_v}\cdot N$ then the bound on the (j,k)-event is obvious by equation (8) (as in case $k \leq j$).

Case 2 (the interesting case). Suppose $|C| \geq \Delta^3\sqrt{p_v}\cdot N$. In this case we use Property (P5) to infer that $|C \cap Y_{n-\ell}| \geq \frac{1}{\Delta}\cdot\frac{|C|}{\sqrt{p_v}N}$. Thus, using equation (9), the probability for the (j,k)-event is at most

$$\begin{aligned}\frac{|C|}{N}\cdot\frac{\#_v(X_{i+k-j}\cap R, Y_{i+k-j}\cap C))}{|C\cap Y_{n-\ell}|} &\leq \frac{|C|}{N}\cdot\frac{5\Delta^3}{|C|/(\Delta\sqrt{p_v}N)}\\ &= 5\Delta^4\cdot\sqrt{p_v}.\end{aligned}$$

The lemma follows. □

Combining Lemmas 4.11 and 4.17, we get the following.

THEOREM 4.18 (advantages in the general case). *Let f be an arbitrary bivariate function and suppose the generic protocol is performed with row and column partitions satisfying Properties* (P0) *through* (P7). *Then, for every value v in the range of f, if one party plays honestly then, no matter how the other player plays, the outcome of the protocol is v with probability at most $O(\log^6(1/p_v) \cdot \max\{q_v, \sqrt{p_v}\})$.*

Proof. Just sum up the bounds for the probabilities of the ℓ^2 events corresponding to the advantage from "nonheavy" strategies (provided by Lemma 4.17) and add the bound on the advantage from heavy strategies provided by Lemma 4.11. (The summation over the strategies is an upper bound, whereas summation over the events corresponding to different column subclasses is exact.) □

We stress that some logarithmic factors (but not all) can be eliminated by a more careful analysis.

4.1.6. Digest of the value-balanced properties. The value-balance properties, referred to in Theorem 4.18, are tabulated in Table 1. Property (P2) is a specialization of Property (P7) for the case $q_v = p_v$ and is not used in the proof of Theorem 4.18 (but rather in the proof of Corollary 4.8). Properties (P2) and (P7) differ from all other value-balance properties in that their definition depends on a standard coloring of a graph induced by the current residual matrix $X_i \times Y_i$. In particular, the sets relevant to these properties in different rounds vary in size. In contrast, we stress that sets relevant to the other properties reduce to about a half with every round. This "irregularity" of Properties (P2) and (P7) introduces difficulties in the subsequent subsections. To compensate for these difficulties, these properties were defined to hold vacuously as long as the residual matrix is "large" (i.e., $\Omega(1/p_v)$). As we pointed out, this convention does not affect the analysis, since Properties (P2) and (P7) are applied only to "small" residual matrices. For similar reasons, Property (P6) which refers to many (i.e., $|Y_i|$) sets which may be very small is also defined to hold vacuously in case the number of sets is much larger than the size of these sets. Note that all other properties either apply to fewer (i.e., $\text{poly}(\ell)$) sets or refer to relatively big sets. Specifically, Properties (P3), (P4), and (P5) apply to $\text{poly}(\ell)$ sets. On the other hand, whenever Properties (P0) and (P1) are applied to many, say M, sets each of these sets has cardinality at least $M/2$ and $(p_v/4) \cdot M$, respectively.

4.2. On the existence of value-balanced partitions. In this subsection we prove the existence of partitions that have all the value-balanced properties used in the previous subsection. We first bound the probability that a random partition is not balanced with respect to a specific set. In the analysis we use an unspecified constant, denoted c_1. The constant δ (in the definition of Δ_v) is determined in terms of c_1 (in fact $\delta = O(c_1)$ will do, $c_1 \geq 2$ suffices for the results of the current subsection and $c_1 \geq 10$ suffices for all the results of the entire section).

LEMMA 4.19. *Let $S \subseteq U$ be finite sets, with $|S| = k$. Then, for every $c_1 > 0$ there exists δ, so that a uniformly selected bipartition of U is Δ_v-biased with respect to S with probability $\geq 1 - (p_v/k)^{c_1}$.*

Proof. We consider two cases corresponding to the two conditions of Definition 4.1. By Chernoff's Bound, the probability that a uniformly selected partition fails condition (1) in Definition 4.1 (with respect to a set S with $k \geq \Delta_v^4$) does not exceed

$$2\exp\{-2(k^{-1/4})^2 \cdot k\} = 2\exp\{-2k^{1/2}\}. \tag{10}$$

TABLE 1
Value-balanced properties (recall $\ell \stackrel{\text{def}}{=} (1/2)\log_2(1/p_v)$*).*

	stated for	Description of the property: the partition approximately halves the number of	Number of sets for $M \times M$ matrix	Applicable for
P0	col	columns	1	all M
P1	col	columns with v-entries in row x, per row	M	all M
P2	col	columns with v-entries in color ϕ, per color	$\leq 2M+1$	$M \leq 2/p_v$
P3	row	heavy rows	1	all M
P4	row	rows of approximate weight 2^{-j}, per $j = 0, \ldots, \ell$	$\leq \ell$	all M
P5	col	columns of a weight class inside a row class, per row class and column subclass	$< 2\ell^2$	all M
P6	row	rows with v-entries in subcolumn, per column and row class (provided residual subcolumn is sufficiently dense)	$M \cdot \ell$	all M
P7	col	columns with v-entries in color ϕ, per color in rectangle	$\leq (2M+1) \cdot \ell^2$	$M \leq 4/p_v$

Using $k \geq (\delta \log_2(1/p_v))^4$, we upper bound equation (10) by

$$\exp\{-k^{1/2}\} \cdot \exp\{(\delta \cdot \log_2(1/p_v))^2\},$$

which for sufficiently large δ (or $1/p_v$) yields the desired bound (of $(p_v/k)^{c_1}$). Similarly, the probability that condition (2) is not satisfied by a random partition is bounded by

$$2\exp\{-2(1/20)^2 \cdot k\} = 2\exp\{-k/200\}. \tag{11}$$

Using $k > \delta \log_2(1/p_v)$ and $\delta \geq 400c_1$, we upper bound equation (11) by

$$\exp\{-k/400\} \cdot \exp\{c_1 \log_2(1/p_v)\},$$

which for sufficiently large δ (or $1/p_v$) yields again the desired bound. □

PROPOSITION 4.20 (existence of value-balance partitions). *Let the generic protocol run for i rounds, using only partitions which satisfy all value-balance properties* w.r.t. all values in *Range*(f). *Let $X_i \times Y_i$ be the residual matrix after these i rounds. Then there exist a row partition* (of X_i) *and a column partition* (of Y_i) *that satisfy all value-balance properties* w.r.t. all values. *Furthermore, for every $v \in$ Range(f), all but a $p_v^{c_1-1}$ fraction of the possible partitions satisfy all v-balance properties.*

Proof. We consider only row partitions, the proof for column partitions being identical. Let $v \in \text{Range}(f)$. For $|X_i| < \Delta_v$ every nontrivial partition will do, so henceforth we assume $|X_i| \geq \Delta_v$. Lemma 4.19 yields an upper bound on the probability that a uniformly chosen partition of X_i violates one of the v-balance properties. For each property, we multiply the number of sets considered by the probability that a uniformly selected bipartition of X_i is not Δ_v-biased with respect to an individual set. An obvious (lower) bound on the size of an individual set considered is Δ_v, but in some cases better lower bounds hold. For each of the eight properties, we prove an upper bound of $p_v^{c_1-1}/8$ on the probability that a uniformly chosen partition violates the property.

- Property (P0) is violated with probability at most $|X_i| \cdot (p_v/|X_i|)^{c_1}$ which can be bounded by $p_v^{c_1-1}/8$.
- Property (P1) is violated with probability at most $|Y_i| \cdot \max_{y \in Y_i}\{(p_v/\#_v(X_i, y))^{c_1}\}$. In case $|Y_i| < \Delta/p_v$, this probability is easily bounded by $(p_v/\Delta_v)^{c_1-1} <$

$p_v^{c_1-1}/8$. Otherwise, we argue as follows. Since Property (P1) was satisfied in previous rounds, it follows (as in Lemma 4.5) that

$$\begin{aligned}\#_v(X_i, y) &\geq 2^{-i-1} \cdot \#_v(X_0, y) \\ &\geq \frac{p_v}{8} \cdot |X_i|\end{aligned}$$

and so Property (P1) is violated with probability at most $|Y_i| \cdot (8/|X_i|)^{c_1}$. Using Property (P0) for the previous rounds we get $|X_i| \geq |Y_i|/4$ and again obtain a bound of $O((p_v/\Delta_v)^{c_1-1}) < p_v^{c_1-1}/8$.

- For Property (P2), we need only consider the case $|X_i| < (2/p_v)$. In this case, Property (P2) is violated with probability at most $(|X_i| + |Y_i| + 1) \cdot (p_v/\Delta_v)^{c_1}$ which is bounded by $O(p_v^{c_1-1}/\Delta_v^{c_1}) < p_v^{c_1-1}/8$. Property (P7) is dealt similarly, but the bound here is $O(p_v^{c_1-1}/\Delta^{c_1-2}) < p_v^{c_1-1}/8$.
- For Property (P6) we need to consider only $j \leq \ell + 1$ and $y \in Y_i$ such that $\#_v(R^{j+\ell} \cap X_i, y) \geq \max\{\Delta_v, (p_v/4\Delta_v) \cdot |Y_i|\}$. Let us denote the set of these pairs by P_i. Then, Property (P6) is violated with probability at most

$$\begin{aligned}\sum_{(j,y)\in P_i} \left(\frac{p_v}{\#_v(R^{j+\ell} \cap X_i, y)}\right)^{c_1} &\leq \left(\frac{p_v}{\Delta}\right)^{c_1-1} \cdot \left(|P_i| \cdot \frac{p_v}{(p_v/4\Delta_v) \cdot |Y_i|}\right) \\ &\leq \left(\frac{p_v}{\Delta}\right)^{c_1-1} \cdot \frac{(\ell+1) \cdot |Y_i|}{|Y_i|/4\Delta} \\ &< \frac{p_v^{c_1-1}}{8}.\end{aligned}$$

- For the remaining properties (i.e., (P3), (P4), and (P5)) we have a total of $O(\log^2(1/p_v))$ sets and so the bound holds easily.

Thus, the probability that a random partition of X_i violates some property with respect to the value v is at most $p_v^{c_1-1}$. The main claim of the proposition follows by summing the bounds obtained for all possible v's and using $c_1 \geq 2$. □

Combining Theorem 4.18 and Proposition 4.20, we get the following.

COROLLARY 4.21 (existence of a protocol meeting the lower bound). *Let f be as in Theorem 4.18. Then, there exists a (deterministic) two-party protocol for computing the function f, so that for every $v \in Range(f)$, if one party plays honestly, then the outcome of the protocol is v with probability at most $O(\log^6(1/p_v) \cdot \max\{q_v, \sqrt{p_v}\})$.*

4.3. Efficient protocols meeting the lower bounds. The protocols guaranteed by Corollary 4.21 are not efficient. In particular, merely specifying the partitions used by the protocol takes space that is exponential in length of the inputs, not to mention that the proof is nonconstructive and that a naive construction would require double exponential time. An efficient implementation of the protocols is achieved by using partitions which can be specified by polynomially many bits. These partitions will not be hardwired into the protocol but rather selected online by the two parties. Namely, at the outset of each step, the parties perform a sampling protocol to select a partition for that step. The partition is specified by an mth degree ($m = \text{poly}(n)$) polynomial over the field $F \stackrel{\text{def}}{=} GF(2^n)$ and a fixed partition of the elements of F into two equal parts F^0 and F^1. For example, suppose polynomial P (over F) is chosen to specify a partition of Y_i, then Y_i^σ is defined as the set of all points $y \in Y_i$ satisfying $P(y) \in F^\sigma$. This plan is materialized via a two-party protocol for sampling these partitions and a proof that, with probability at least $1 - P_v$, every partition

selected (for the generic protocol) by the sampling protocol satisfies all v-balance properties. To this end we first bound the probability that, for an appropriately chosen $m = \text{poly}(n)$, a random mth degree polynomial induces a partition that fails to satisfy some v-balance properties. Next, we present a two-party protocol for sampling l-bit strings and bound the advantage of each party towards any set as a function of the density of that set.

Terminology. Partitions induced by $(\delta n)^4$-degree polynomials are hereafter called *polynomial partitions*. We modify these partitions so that they are never trivial (e.g., by replacing each trivial partition by a fixed nontrivial partition). Recall that Property (P0) forbids trivial partitions, except if the universe is a singleton. The modification is introduced to guarantee this.

4.3.1. Bounding the probability of nonbalanced polynomial partitions. We start by bounding the probability for a random polynomial partition to fail some v-balance property.

LEMMA 4.22. *For every $c_1 > 0$ there exists δ, so that for every set S of cardinality k, a uniformly selected polynomial partition is not Δ_v-biased with respect to S with probability at most $(p_v/k)^{c_1}$.*

Proof. The modification described in the terminology (above) can only decrease the probability that a partition is not Δ-biased (w.r.t. any set S). Thus, it suffices to analyze the distribution of unmodified polynomial partitions.

A $2t$th moment argument easily shows that if $x_1, x_2, \ldots, x_k$ are m-wise independent random variables uniformly distributed in $\{0,1\}$ then $\text{Prob}(|\sum_{i=1}^{k} x_i - \frac{k}{2}| > B) < (\frac{\sqrt{k}t}{B})^{2t}$, for every $t \leq m/2$. Therefore, the probability for a uniformly chosen polynomial partition to fail condition (1) in Definition 4.1 does not exceed

$$\left(\frac{\sqrt{k}\cdot t}{k^{3/4}}\right)^{2t} = \left(\frac{t}{k^{1/4}}\right)^{2t} \tag{12}$$

for any $t \leq (\delta n)^4/2$. We now use equation (12) with two different settings for t. First we set $t = \Delta_v/2$ (since $p_v \geq 2^{-n}$, it follows that $\Delta_v \leq \delta \cdot 2n$ and this t is indeed smaller than $(\delta n)^4/2$) and using $k \geq \Delta^4$, we bound equation (12) by

$$\left(\frac{\Delta_v/2}{\Delta_v}\right)^{\Delta_v} = p_v^{\delta} < p_v^{2c_1},$$

where the last inequality comes from $\delta \geq 2c_1$. Secondly, we set $t = 8c_1$, and bound equation (12) by

$$\left(\frac{8c_1}{k^{1/4}}\right)^{16c_1} = \left(\frac{(8c_1)^8}{k^2}\right)^{2c_1} < \frac{1}{4}\cdot k^{-2c_1},$$

where we have used $k \geq \Delta^4 \geq 4 \cdot (8c_1)^8$. Multiplying these two bounds, we bound equation (12) by

$$\sqrt{p_v^{2c_1} \cdot \frac{k^{-2c_1}}{4}} = \frac{1}{2}\cdot (p_v/k)^{c_1}$$

as desired. To bound the probability for failure in condition (2), note that for $k \leq \Delta^4$ we have, $k \leq (\delta n)^4$ (as previously observed $\Delta_v \leq \delta n$). Thus, a uniformly selected

polynomial partition splits k elements exactly as a totally random partition and so the bound obtained for this case (i.e., for $k \leq \Delta^4$) in Lemma 4.19 holds also here. □

PROPOSITION 4.23 (polynomial-partition satisfy value-balance properties). *Fix $v \in Range(f)$, and consider an execution of the generic protocol with uniformly selected polynomial partitions. Let π_i be the probability that the first failure of some v-balance property occurs on the ith round. Then,*

$$\sum_{i\geq 1} \sqrt[4]{\pi_i} \leq O(\Delta_v \cdot p_v).$$

The mysterious choice of the 4th roots will be clarified when we apply the proposition (in the proof of Theorem 4.27).

Proof. It suffices, of course, to consider only row partitions. Let $\pi_{i,t}$ be the probability that our first failed row partition occurred in round i and that Property (P_t) was violated (for some $0 \leq t \leq 7$ and $i \geq 1$). Clearly,

$$\begin{aligned}\sum_{i\geq 1} \sqrt[4]{\pi_i} &\leq \sum_{i\geq 1} \sqrt[4]{\sum_{t=0}^{7} \pi_{i,t}} \\ &\leq \sum_{t=0}^{7} \sum_{i\geq 1} \sqrt[4]{\pi_{i,t}}.\end{aligned}$$

So it remains to bound, $\sum_{i\geq 1} \sqrt[4]{\pi_{i,t}}$, for each $t = 0, \ldots, 7$. Analogously to the proof of Proposition 4.20, we use Lemma 4.22 to upper bound the probability that a uniformly chosen polynomial partition violates one of the v-balance properties. (Again, for each property, we multiply the number of sets considered by the probability that a uniformly selected polynomial partition is not Δ_v-biased with respect to an individual set. An obvious (lower) bound on the size of an individual set considered is Δ_v, but in some cases better lower bounds hold). We now assume $c_1 \geq 10$.

- We upper bound the probability that Property (P0) is violated for the first time in the ith round by $|X_{i-1}| \cdot (p_v/|X_{i-1}|)^{c_1}$. Letting $x_j := |X_j|$, we have

$$\pi_{i,0} \leq x_{i-1} \cdot (p_v/x_{i-1})^{c_1}, \tag{13}$$

$$\text{where } x_j \geq \max\{\Delta, |X_0|/2^{j-1}\}, \tag{14}$$

  where the lower bound on x_j follows, since Property (P0) held in the previous rounds. Furthermore, if Property (P0) held in all first n rounds, then $|X_n| \leq \Delta$ and henceforth every nontrivial partition satisfies all properties vacuously. Therefore,

$$\begin{aligned}\sum_{i\geq 1} \sqrt[4]{\pi_{i,0}} &= \sum_{i=1}^{n} \sqrt[4]{\pi_{i,0}} \\ &\leq \sum_{i=1}^{n} \sqrt[4]{x_{i-1} \cdot \left(\frac{p_v}{x_{i-1}}\right)^{c_1}} \\ &< \sum_{i=1}^{n} \frac{p_v}{x_{i-1}} \\ &< p_v \cdot \sum_{i=1}^{n} \frac{2^{i+2}}{2^n},\end{aligned}$$

where the last inequality uses the lower bounds for the x_j's. It follows that $\sum_{i\geq 1}\sqrt[4]{\pi_{i,0}} = O(p_v)$.

- Adopting the analysis in the proof of Proposition 4.20, we know that the probability that the first failure is with Property (P1) in round i is at most $4|X_{i-1}|\cdot(p_v/|X_{i-1}|)^{c_1}$. Using the same analysis as above, we conclude $\sum_{i\geq 1}\sqrt[4]{\pi_{i,1}} = O(p_v)$.
- For Properties (P2) and (P7), we need only consider rounds i so that $|X_i| < (2/p_v)$. Using the analysis in the proof of Proposition 4.20, we bound the probability that the partition in such a round violates Property (P2) (resp., (P7)) by $O(p_v^{c_1-1}/\Delta_v^{c_1})$ (resp., $O(p_v^{c_1-1}/\Delta^{c_1-2})$). The bound on $\sum_{i\geq 1}\sqrt[4]{\pi_{i,t}}$, for $t = 2, 7$, follows, since there are at most Δ such rounds.
- Following the analysis in the proof of Proposition 4.20, we consider for Property (P6) only $j \leq \ell+1$ and $y \in Y_i$ such that $\#_v(R^{j+\ell}\cap X_i, y) \geq \max\{\Delta_v, (p_v/4\Delta_v)\cdot|Y_i|\}$. Let us denote the set of these pairs by P_i. The probability that our first violation is on round i and Property (P6) is being violated, is at most

$$\begin{aligned}\sum_{(j,y)\in P_i}\left(\frac{p_v}{\#_v(R^{j+\ell}\cap X_i, y)}\right)^{c_1} &\leq |P_i|\cdot\left(\frac{p_v}{\Delta}\right)^{(c_1+1)/2}\cdot\left(\frac{p_v}{(p_v/4\Delta_v)\cdot|Y_i|}\right)^{(c_1+1)/2}\\ &\leq ((\ell+1)\cdot|Y_i|)\cdot\left(\frac{p_v}{\Delta}\right)^4\cdot\left(\frac{4\Delta}{|Y_i|}\right)^5\\ &< \left(\frac{\Delta_v\cdot p_v}{|Y_i|}\right)^4.\end{aligned}$$

  Using the same analysis as for Property (P0), we obtain $\sum_{i\geq 1}\sqrt[4]{\pi_{i,6}} < \Delta_v\cdot p_v$.
- For the remaining properties (i.e., (P3), (P4), and (P5)) we have a total of $O(\log^2(1/p_v))$ sets and so we can handle each of these sets separately. Consider, for example, the set $R^{j+\ell}$ from the definition of Property (P4). The row partition of round $i+1$ violates the balance property on this set with probability at most $(\frac{p_v}{|R^{j+\ell}\cap X_i|})^{c_1}$. Setting $x_i \stackrel{\text{def}}{=} |R^{j+\ell}\cap X_i|$, we can apply the same analysis as applied to equation (13), except that here we use Property (P4) for the previous rounds. The desired bound for $\sum_{i\geq 1}\sqrt[4]{\pi_{i,t}}$ follows, for $t = 3, 4, 5$.

Having shown that $\sum_{i\geq 1}\sqrt[4]{\pi_i} < \Delta_v\cdot p_v$, for each $t = 0,\ldots,7$, the proposition follows. □

4.3.2. A protocol for string sampling. We now present a two-party protocol for sampling l-bit strings and bound the advantage of each party towards any set as a function of the set's density. The protocol is a simplification of the protocol for computing a function. The parties proceed in l rounds. In each round one party should select a pseudorandom partition of the residual sample space and the other party should flip a coin to select a side of this partition. In the next round the parties switch roles. All partitions selected by each party must divide the residual space into two sets of equal cardinality. Specifically, the partition is defined by a linear combination of the bits in the representation of the sample point. Following is the code of the protocol (the parties are called P_0 and P_1).

Round i:

- $P_{i \bmod 2}$ uniformly selects an l-dimensional binary vector v_i, which is linearly independent of the vectors used in previous rounds, and sends v_i to the other party.
- $P_{(i+1) \bmod 2}$ uniformly selects $\sigma_i \in \{0,1\}$ and sends it to the other party.

Intuition: The residual sample space after round i consists of all l-dimensional binary vectors x so that $< x, v_j > = \sigma_j$ for every $j \le i$ ($< \cdot, \cdot >$ is mod-2 inner product, and this residual set is an affine subspace).

PROPOSITION 4.24 (analysis of the two-party sampling protocol). *Let $S \subseteq \{0,1\}^l$ be arbitrary and let $p \stackrel{\text{def}}{=} |S|/2^l$. If one of the parties that participate in the above protocol plays honestly, then the probability for the protocol's outcome to be in S is at most $O(p^{\frac{1}{4}})$.*

Proof. Let U_i denote the residual sample space after round i; namely,

$$U_i \stackrel{\text{def}}{=} \{x :< x, v_j > = \sigma_j \ \ \forall j \le i\}.$$

Let $S_i \stackrel{\text{def}}{=} S \cap U_i$ denote the residual target set ($U_0 = \{0,1\}^l$ and $S_0 = S$). We want to consider the cardinality of S_i as i grows (i.e., the execution proceeds) and treat differently "small" and "large" S_i. For "small" S_i we bound the probability of hitting S_i as $|S_i|$ times the probability of hitting any specific element. If S_i is "large," then with sufficiently high probability $|S_{i+1}| \approx |S_i|/2$ and hence the density, $|S_i|/|U_i|$, is approximately preserved. Details follow.

The following three claims do not depend on the residual sample space U_i. Thus, S_i (the residual target set after i rounds) can be considered fixed, too.

Claim 4.24.1. If the $(i+1)$st partition is chosen by an honest player, then, with probability at least $1 - |S_i|^{-\frac{4}{5}}$:

$$\frac{|S_i|}{2} - |S_i|^{\frac{9}{10}} < |S_{i+1}| < \frac{|S_i|}{2} + |S_i|^{\frac{9}{10}},$$

regardless of the choice of σ_{i+1}.

Proof. By hypothesis, v_{i+1} is uniformly selected among the vectors which are linearly independent of $v_1, \ldots, v_i$. Instead, let us select v_{i+1} uniformly at random from the entire space Z_2^l. The additional partitions come from v_{i+1} in the linear span of $(v_1, \ldots, v_i)$, and thus induce a trivial partition on U_i, so the modified partitioning procedure is only less likely to yield good partitions.

We show that with very high probability, even the partition induced by a uniformly chosen vector is quite balanced. For any $\sigma \in \{0,1\}$, we consider random variables ζ_s, $(s \in S_i)$ where $\zeta_s = 1$ if $< s, v_{i+1} >= \sigma$ and 0 otherwise. Since v_{i+1} is selected uniformly, each ζ_s is uniformly distributed in $\{0,1\}$. Furthermore, these random variables are pairwise independent, as long as $|U_i| \ge 2$ (i.e., the protocol did not terminate). Thus, we have

$$\text{Prob}\left(\left|\sum_{s \in S_i} \zeta_s - \frac{|S_i|}{2}\right| \ge |S_i|^{\frac{9}{10}}\right) < \frac{|S_i|}{|S_i|^{2 \cdot \frac{9}{10}}}$$

and the claim follows. □

On the other hand, the following claim is obvious.

Claim 4.24.2. If σ_{i+1} is selected by an honest player, then the expected cardinality of S_{i+1} is $|S_i|/2$.

The probability of hitting S_i is bounded by $|S_i|$ times the probability of hitting any specific element of S_i, so we have the following.

Claim 4.24.3. With the above notation, the probability that the output of the protocol is in S (or, equivalently, in S_i) does not exceed $|S_i| \cdot 2^{-(l-i-1)/2}$.

Proof. Clearly $|U_i| = 2^{l-i}$ and there remain $r \stackrel{\text{def}}{=} l - i$ rounds to termination, of which σ will be chosen by an honest player at least $\lfloor r/2 \rfloor$ times. Any $s \in S_i$

survives each such round with probability $\frac{1}{2}$, and is the output with probability at most $\cdot 2^{-\lfloor r/2 \rfloor}$, as claimed. □

In case $|S| < p^{-\frac{1}{2}}$ the proposition follows by using Claim 4.24.3; namely, the probability for output in S is bounded by

$$\begin{aligned} |S_0| \cdot 2^{-l/2} &= \sqrt{|S| \cdot \frac{|S|}{2^l}} \\ &= \sqrt{|S| \cdot p} \\ &< p^{\frac{1}{4}}. \end{aligned}$$

So in what remains we consider the case $|S| \geq p^{-\frac{1}{2}}$. Let the protocol be executed for $t \stackrel{\text{def}}{=} \log_2 |S| - \frac{1}{2}\log_2(1/p) \geq 0$ rounds. In the rest of the proof we essentially show that, at this stage, $|S_t| \approx p^{-\frac{1}{2}}$. Using Claim 4.24.3 at this point, we obtain (again) the upper bound of $|S_t| \cdot 2^{-(l-t)/2} = p^{\frac{1}{4}}$ (using $l - t = l - \log_2 |S| + \frac{1}{2}\log_2(1/p) = (1+\frac{1}{2}) \cdot \log_2(1/p)$).

We assume, without loss of generality, that the honest party picks the partitions at the even rounds. Also, there is no loss in assuming that his opponent plays a pure (i.e., deterministic) strategy: since the honest party's strategy is fixed, the adversary's optimal move maximizes his expected payoff. On even-numbered rounds he selects one side of a partition presented by the honest player, while on round $2i+1$ he selects a partition that is determined by a function Π_i. Formally, each of his moves is a function of the history of the execution, but this whole history is encoded by the current residual sample space. Thus, we may view each Π_i as a mapping $\Pi_i : 2^U \mapsto 2^U$, where U_{2i-2}, the residual sample space after $2i-2$ rounds is partitioned into $(\Pi_i(U_{2i-2}), U_{2i-2} - \Pi_i(U_{2i-2}))$. Having fixed the adversary's strategy, the residual sample space after j rounds, U_j is a well-defined random variable. The following two sequences of random variables, depend now only on the coin tosses of the honest party:

1. π_i is the cardinality of $S \cap \Pi_i(U_{2i-2})$, for $i \geq 1$;
2. ζ_j is the cardinality of $S \cap U_j$, for $j \geq 0$ (where, $\zeta_0 = |S|$ is constant).

The following facts are immediate by the definitions and Claims 4.24.1 and 4.24.3.

Claim 4.24.4. For every $i \geq 1$,

1. (*effect of round* $2i-1$: adversary presents partition)
   $\text{Prob}(\zeta_{2i-1} = \pi_i) = \text{Prob}(\zeta_{2i-1} = \zeta_{2i-2} - \pi_i) = \frac{1}{2}$.
2. (*effect of round* $2i$: adversary selects side)
   $|\zeta_{2i} - \frac{\zeta_{2i-1}}{2}| < \zeta_{2i-1}^{\frac{9}{10}}$ with probability at least $1 - \zeta_{2i-1}^{-\frac{4}{5}}$. Always $0 \leq \zeta_{2i} \leq \zeta_{2i-1}$.
3. (*termination*: as a function of the situation after $t \stackrel{\text{def}}{=} \log_2 |S| - \frac{1}{2}\log_2(1/p)$ rounds)
   The protocol terminates with output in S with probability at most
   $$\text{Exp}(\zeta_t) \cdot 2^{-(l-t)/2} = \text{Exp}(\zeta_t) \cdot p^{3/4}$$
   the expectation being over the coin tosses of the honest player in the first t rounds.

In proving item (3), use $\text{Exp}(\zeta_t \cdot 2^{-(l-t)/2}) = \text{Exp}(\zeta_t) \cdot 2^{-(l-t)/2}$ and $l - t = l - \log_2 |S| + \frac{1}{2}\log_2(1/p) = (1 + \frac{1}{2}) \cdot \log_2(1/p)$. It remains to use items (1) and (2) in order to prove the following.

Claim 4.24.5. Let $t \stackrel{\text{def}}{=} \log_2 |S| - \frac{1}{2}\log_2(1/p)$ and suppose $t \geq 0$. Then

$$\text{Exp}(\zeta_t) = O(p^{-1/2})$$

the expectation being over the coins tossed by the honest player in the first t rounds.

Proof. Using item (2) of Claim 4.24.4, we obtain

$$\begin{aligned}\mathrm{Exp}(\zeta_{2i+2}) &\leq \mathrm{Exp}\left(\frac{\zeta_{2i+1}}{2} + \zeta_{2i+1}^{\frac{9}{10}} + \zeta_{2i+1}^{-\frac{4}{5}} \cdot \zeta_{2i+1}\right) \\ &\leq \mathrm{Exp}\left(\frac{\zeta_{2i+1}}{2} + 2 \cdot \zeta_{2i+1}^{\frac{9}{10}}\right).\end{aligned}$$

On the other hand, using item (1) of Claim 4.24.4, we obtain both

$$\mathrm{Exp}(\zeta_{2i+1}) = \frac{1}{2} \cdot \mathrm{Exp}(\zeta_{2i})$$

and

$$\mathrm{Exp}(\zeta_{2i+1}^{\frac{9}{10}}) = \frac{1}{2} \cdot \mathrm{Exp}(\pi_i^{\frac{9}{10}}) + \frac{1}{2} \cdot \mathrm{Exp}((\zeta_{2i} - \pi_i)^{\frac{9}{10}}).$$

Combining the three (in)equalities, we get

$$\begin{aligned}\mathrm{Exp}(\zeta_{2i+2}) &\leq \frac{1}{4} \cdot \mathrm{Exp}(\zeta_{2i}) + \mathrm{Exp}(\pi_i^{\frac{9}{10}}) + \mathrm{Exp}((\zeta_{2i} - \pi_i)^{\frac{9}{10}}) \\ &< \frac{1}{4} \cdot \mathrm{Exp}(\zeta_{2i}) + 2 \cdot \mathrm{Exp}(\zeta_{2i}^{\frac{9}{10}}).\end{aligned}$$

For $0 < \alpha < 1$, the function x^α over $x \geq 0$ is concave, so we may apply Jensen's inequality, and conclude

$$\mathrm{Exp}(\zeta_{2i+2}) < \frac{1}{4} \cdot \mathrm{Exp}(\zeta_{2i}) + 2 \cdot \mathrm{Exp}(\zeta_{2i})^{\frac{9}{10}}.$$

Setting $z_i \stackrel{\text{def}}{=} \mathrm{Exp}(\zeta_{2i})$, a minor adaptation of Claim 4.4 yields $\mathrm{Exp}(\zeta_t) = O(\frac{\zeta_0}{2^t})$. Recall now that $t = \log_2 |S| - \frac{1}{2}\log_2(1/p))$ and $\zeta_0 = |S|$, to conclude the claim. □

The proposition follows. □

Remark 4.25. The bound provided in Proposition 4.24 is not tight. Yet, it suffices for the purpose of sampling partitions in the generic protocol (see the proof of Theorem 4.27). Much better protocols can be obtained—see Theorem 4.28. These (more complex) sampling protocols use the above protocol and the bound from Proposition 4.24 as a bootstrapping step. In our best sampling protocol, if one party plays honestly, the probability for the protocol to land in an element of any set of density p does not exceed $O(\sqrt{p})$.

Remark 4.26. Our two-party sampling protocol is very similar to *interactive hashing*, a protocol that was discovered independently by Ostrovsky, Venkatesan, and Yung [20] (see Naor et. al. [18]). However, in interactive hashing one party always picks the partition and the other always chooses the side. Also, interactive hashing terminates after $l - 1$ (rather than l) rounds. Interactive hashing was invented for completely different purposes and consequently its analysis, as in [18] (and subsequent studies), is very different from what appears above. Interactive hashing was used for implementing various types of commitment protocols (cf. [20, 18, 21, 10]).

4.3.3. The main result. Combining Propositions 4.23 and 4.24 with Theorem 4.18, we get the following.

THEOREM 4.27 (efficient protocol meeting the lower bound). *There exists a (generic) two-party protocol, for evaluating an arbitrary bivariate function f. This protocol is performed by a pair of uniform probabilistic polynomial-time programs with a single oracle call to the function f and satisfies the following properties:*

- *If both parties play honestly and their inputs are x and y respectively, then the output is $f(x,y)$.*
- *For every value v in the range of f, if one party plays honestly then the outcome of the protocol is v with probability at most*

$$O(\log^6(1/p_v)\cdot\max\{q_v,\sqrt{p_v}\}).$$

Furthermore, in case $q_v = p_v$, this bound can be improved to $O(\sqrt{p_v})$.

Proof. The protocol is an implementation of the generic protocol where the partitions are determined by poly(n)-degree polynomials that are selected using the sampling protocol described above. This proves the first item. For the second item we consider the event in which during the execution of the protocol (with at least one party being honest) a partition was selected which does not satisfy all v-balanced properties. Using Propositions 4.23 and 4.24, the probability of this event is $O(\Delta_v \cdot p_v)$. (Here we use the fact that Proposition 4.23 bounds the sum of the fourth root of the density of "bad" partitions.) In the complementary case, when every partition that is used satisfies all v-balance properties, Theorem 4.18 applies, and the main part of the second item follows.

A bound of $O(\sqrt{p_v}\log^2(1/p_v))$ for the special case of $q_v = p_v$ can be obtained by using Corollary 4.8 instead of Theorem 4.18. The better bound of $O(\sqrt{p_v})$ requires a slightly more careful analysis that we turn to perform.

We slightly change the classification of rounds as appearing in the motivating discussion (subsection 4.1). We first consider the situation after $i \stackrel{\text{def}}{=} n - \log_2(1/p_v) - 4\log_2\Delta_v$ rounds. Following the ideas in the proof of Lemma 4.5 (and using Proposition 4.23 and 4.24), we first observe that, with probability $\geq 1 - p_v$, the number of v-entries in each row (column) of the residual matrix is at most $2\cdot\Delta_v^4$ (i.e., $\#_v(x,Y_i) \leq 2\Delta_v^4$, $\forall x \in X_i$). (Here and below the probability space is comprised of runs of the generic protocol in which polynomial partitions are selected using the sampling protocol of Proposition 4.24.) Next, we consider the situation after an additional $\ell \stackrel{\text{def}}{=} \frac{1}{2}\log_2(1/p_v)$ rounds. Using similar ideas (this time following Lemma 4.7), we conclude that, with probability $\geq 1 - p_v$, the total number of v-entries in the entire residual matrix, is at most $(4\Delta_v^4+1)\cdot 2\Delta_v^4 < 9\Delta_v^8$ (i.e., $\#_v(X_{i+\ell},Y_{i+\ell}) < 9\Delta_v^8$). Furthermore, with probability at least $1-p_v$, the residual matrix at this point is of size approximately $\frac{\Delta_v^4}{\sqrt{p_v}}$ by $\frac{\Delta_v^4}{\sqrt{p_v}}$. In the original analysis, we did not try to argue that the number of v-entries in each row/column decreases during these additional ℓ rounds. But this is most likely to be the case as shown below.

Claim 4.27.1. There exists a constant c so that with probability at least $1-p_v$, after $i+\ell = n - \frac{1}{2}\log_2(1/p_v) - 4\log_2\Delta_v$ rounds, there are at most c v-entries in each residual row (resp., column) (i.e., $\#_v(x,Y_{i+\ell}) \leq c$, $\forall x \in X_{i+\ell}$).

Proof. We consider again these additional ℓ rounds, assuming that previously (i.e., after i rounds) each residual row/column contains at most $2\Delta_v^4$ v-entries. We want to bound, for each individual row $x \in X_i$, the probability that $\#_v(x,Y_{i+\ell}) > c$. Say that a column partition is *good* if either there are fewer than c v-entries in the x-row, or each side of the partition contains at least one-third of these entries. (In a *good* round, a good column partition is performed). A *uniformly* selected polynomial partition fails to be good with probability that is exponentially small in the number of v-entries, since at this point, the degree of the polynomials that determine the partition exceeds the number of v-entries in row x. However, the polynomial partitions are selected using the sampling protocol of Proposition 4.24. As Proposition 4.24 states, the same remains valid also when using the sampling algorithm to select the partitions (at the

cost of a different constant in the exponent). Therefore, there exists a constant c so that, as long as row x has more than c v-entries, the next round is good with probability at least 16/17 (a great underestimate for all but the very last rounds). On the other hand, if we go through at least $t \stackrel{\text{def}}{=} \log_{3/2}(2\Delta_v^4)$ *good* rounds, then row x has at most c v-entries. Thus $\#_v(x, Y_{i+\ell}) > c$ only if fewer than $t \ll \ell = \frac{1}{2}\log_2(1/P_v)$ out of the last ℓ rounds are good, and the probability of this event is bounded above by

$$\binom{\ell}{t} \cdot (1/17)^{\ell-t} < (1/16)^{(1+\epsilon)\cdot\ell} = p_v^{(1+\epsilon)\cdot 2},$$

where $\epsilon > 0$ is some small constant, the inequality follows by $t = o(\ell)$ and the equality uses the definition of ℓ. Summing over all possible $x \in X_i$, the claim follows. □

Combining Claim 4.27.1 with the discussion which precedes it, we conclude that after $i+\ell = n - \frac{1}{2}\log_2(1/p_v) - 4\log_2 \Delta_v$ rounds, with very high probability, the residual matrix contains at most $9\Delta_v^8$ entries of value v with at most c such entries in any row or column. Since we are seeking an $O(\sqrt{p_v})$ bound, we can and will ignore those rare runs (of probability $O(p_v)$), for which this is not the case. Proceeding analogously to subsection 4.1, we could consider the situation after another $r = 4\log_2 \Delta_v$ rounds and bound by p_v the probability that after a total of $i + \ell + r = n - \frac{1}{2}\log_2(1/p_v)$ rounds the residual submatrix contains more than Δ_v entries of value v. This would yield a bound of $O(\Delta_v \cdot \sqrt{p_v})$ on the influence towards v. To obtain the better bound claimed above, we observe that it suffices to bound the *expected number* of v-entries in the residual matrix (rather than bounding the probability that too many v-entries remain). Specifically, we consider a standard coloring of the v-entries after $i+\ell$ rounds. This coloring uses at most $2 \cdot c + 1$ colors. Fixing one of these colors, we consider the next $r \stackrel{\text{def}}{=} 4\log_2 \Delta_v$ rounds, and bound the expected number of the remaining v-entries. A *diagonal* is a set of entries in a matrix that has no more than a single element in common with any row/column.

Claim 4.27.2. Consider a diagonal D of at most $9\Delta_v^8$ entries in the residual matrix $(X_{i+\ell} \times Y_{i+\ell})$. Then the expected number of entries from D in the residual matrix $X_{i+\ell+r} \times Y_{i+\ell+r}$ is $O(1)$.

Proof. It suffices to analyze a process in which $2r = 8\log_2 \Delta_v$ polynomial partitions, selected by the sampling protocol of Proposition 4.24, are applied to a space containing $9 \cdot \Delta_v^8$ elements so that after selecting each partition we proceed with the side containing more elements. Our claim is that the expected number of elements after applying these $2r$ partitions is $O(1)$. To prove this claim, let us consider first what happens after applying a single partition. Namely, let S be a subset (of some universe) and ζ be a random variable representing the number of S-elements in the S-heavier side (i.e., the side containing more S-elements) of a partition, selected by the sampling protocol. Clearly,

$$\text{Exp}(\zeta) < \left[\frac{|S|}{2} + |S|^{3/4}\right] + \text{Prob}\left(\zeta > \frac{|S|}{2} + |S|^{3/4}\right) \cdot |S|.$$

For a uniformly selected polynomial partition the probability that the S-heavy side contains more than $|S|/2 + |S|^{3/4}$ elements of S is exponentially small in $\sqrt{|S|}$ and by Proposition 4.24 the same holds (with a smaller constant in the exponent) when the polynomial partition is selected by the sampling protocol. Thus, $\text{Exp}(\zeta) < \frac{|S|}{2} + |S|^{3/4} + O(1)$. Hence, we have a sequence of random variables, $\zeta_0, \zeta_1, \ldots, \zeta_{2r}$, so that $\zeta_0 < 9\Delta_v^8$ and $\text{Exp}(\zeta_i | \zeta_{i-1} = s) < \frac{s}{2} + s^{3/4} + O(1)$, for $i = 1, \ldots, 2r$. Manipulating the

expectation operators (as in the proof of Claim 4.24.5), we conclude that $\mathrm{Exp}(\zeta_{2r}) = O(1)$ and the current claim follows. □

Combining Claims 4.27.1 and 4.27.2, we conclude that with probability $1 - p_v$ we reach round $i + \ell + r = n - \frac{1}{2}\log_2(1/p_v)$ with an expected number of $O(1)$ entries of value v. Using the analysis of Corollary 4.8 (corresponding to stage 3 in the motivating discussion) we establish the claimed $O(\sqrt{p_v})$ bound and the theorem follows. □

As stated in Remark 4.25, we have sampling protocols that improve on Proposition 4.24. This can be done either directly (with the techniques used in proving Theorem 4.27) or by applying Theorem 4.27 to any function f with $q_v = 2^{-l}$ $(\forall v \in \{0,1\}^l)$. In either case, the resulting sampling protocols use the simple sampling protocol (and the bound presented in Proposition 4.24 as a bootstrapping step).

THEOREM 4.28 (a better two-party sampling protocol). *There exists a protocol for sampling $\{0,1\}^l$ that is performed by a pair of uniform probabilistic polynomial-time programs, so that: For every $S \subseteq \{0,1\}^l$ of density p, if one party plays honestly, the outcome of the protocol is in S with probability at most $O(\sqrt{p})$.*

Proof (using the second alternative). Let $f : \{0,1\}^n \times \{0,1\}^n \mapsto \{0,1\}^l$ satisfy $q_v = 2^{-l}$ for every $v \in \{0,1\}^l$. For example, $f(x,y) = x + y \bmod 2^l$, where x and y are viewed as residues mod 2^n (and $n \geq l$, say $n = l$). An honest party is supposed to select its input uniformly in $\{0,1\}^n$ and to invoke the protocol of Theorem 4.27. The current theorem follows from the (furthermore part of) Theorem 4.27, by considering the indicator function $\chi_S(v) = 1$ if $v \in S$ (and $\chi_S(v) = 0$ otherwise). Namely, we consider the function $g(x,y) \stackrel{\mathrm{def}}{=} \chi_S(f(x,y))$ and take advantage of the fact that the protocol in Theorem 4.27 is generic (i.e., determines a pair of inputs (x,y) for the function independently of the function). □

5. Towards the multiparty case. We believe that the ideas developed in the two-party case will prove useful also for the multiparty case. However, even the problem of computing a 3-argument function by a 3-party protocol in the presence of one dishonest party is much more involved than the problem of computing a bivariate function by a 2-party protocol, as in the previous section. A natural extension of our two-party protocol is to let each round consist of three steps (rather than two) and refer to three partitions of the three residual input spaces. In each step, a predetermined party announce in which side of the partition its input lies, and by doing so makes its residual input space smaller. We believe that this (generic) protocol when used with random partitions, nearly minimizes the advantage of any dishonest party, regardless of the function that is being computed. We also believe that this protocol nearly minimizes the advantage of any coalition of two dishonest players. However, this seems to require a much more complex analysis, and additional parameters of the function need to be taken into account. In particular, the advantage of a single adversary towards a value v depends not only on the density of v-entries in the entire function (denoted p_v above) and on the density of v-entries in the function restricted by the best input (denoted q_v). For example, a single party can influence any protocol for computing the function $f(x,y,z) = x + y + z \bmod N$ to produce output 0 (or any other residue mod N) with probability $N^{-2/3}$ (and the generic protocol can be shown to bound the advantage of a dishonest party to about this value). On the other hand, a single party can influence any protocol for computing the function $g(x,y,z) = x + y \bmod N$ to produce output 0 with probability $N^{-1/2}$ (and again the generic protocol meets this bound). However, both functions have the same $p_v = q_v = 1/N$.

Another difficulty which arises in the context of multiparty protocols is that, when the number of parties is large, we cannot afford to let the parties reveal information

in a predetermined order (as in the two-party case and the three-part case above). This difficulty is best demonstrated in the special case where each input is one bit (i.e., $Domain(f) = \{0,1\} \times \{0,1\} \cdots \times \{0,1\}$). Here, the influence of parties which are last to reveal their input is more substantial than the influence of parties which reveal their input first. This calls for choosing a random permutation to determine the order of playing. Thus, the role of a sampling protocol in the multiparty case is more fundamental than in the two-party situation. (Recall that in the two-party protocols, sampling was introduced only for increased efficiency.)

5.1. A multiparty sampling protocol. In this paper we confine ourselves to the presentation of an efficient fault-tolerant multiparty sampling protocol.

THEOREM 5.1 (multiparty sampling protocol). *There exists an m-party sampling protocol that is performed by m (identical) uniform probabilistic polynomial-time programs, so that: For every set $S \subseteq \{0,1\}^l$, if $m-t$ parties play honestly, then the outcome of the protocol is in S with probability at most $O(\log(1/p) \cdot p^{1-O(\frac{t}{m})})$, where $p \stackrel{\text{def}}{=} |S|/2^l$.*

Our proof of Theorem 5.1 adapts the ideas used in Theorem 4.28 to the multiparty context. Namely, our protocol uses partitions which are in turn selected by a lower quality sampling protocol. Specifically, the protocol proceeds in l rounds. In each round, the m parties first select at random (using a simpler sampling protocol) a $\text{poly}(n \cdot m)$-degree polynomial specifying a partition of the residual sample space, and next use the collective coin tossing protocol of Alon and Naor [1] to choose one side of this partition. The sampling protocol used to choose $\text{poly}(nm)$-degree polynomials is similar except that the partitions are specified by linear transformations (as in the protocol of Proposition 4.24). These linear transformations are selected using a trivial sampling protocol which consists of selecting each bit individually by the collective coin tossing protocol of Alon and Naor [1].

We prefer an alternative presentation of our proof, in which the construction of multiparty sampling protocols is reduced to the construction of sampling algorithms that use an SV-source as their source of randomness. Recall that an *SV-source with parameter $\gamma \geq \frac{1}{2}$* (cf. [22]) is a sequence of Boolean random variables, $X_1, X_2, \ldots$, so that for each i and every $\alpha \in \{0,1\}^i$ and every $\sigma \in \{0,1\}$:

$$\text{Prob}(X_{i+1} = \sigma | X_1 \cdots X_i = \alpha) \leq \gamma.$$

Theorem 5.1 follows from the next proposition.

PROPOSITION 5.2 (sampling with an SV-source). *For every constant γ, $\frac{1}{2} \leq \gamma < \frac{1}{\sqrt{2}}$, there exist a probabilistic polynomial-time algorithm, A_1, which on input 1^n uses any SV-source with parameter γ for its internal coin tosses and satisfies, for every sufficiently large n and every set $S \subseteq \{0,1\}^n$,*

$$\text{Prob}(A_1(1^n) \in S) = O(\log(1/p) \cdot p^{\log_2(1/\gamma)}),$$

where $p \stackrel{\text{def}}{=} \frac{|S|}{2^n}$, and the probability is taken over an arbitrary SV-source with parameter γ.

In particular, for $\gamma = \frac{1}{2}(1+\epsilon)$, we have $\log_2(1/\gamma) = 1 - \log_2(1+\epsilon) \geq 1 - \frac{1}{\ln 2} \cdot \epsilon$. Thus, observing that the Alon–Naor protocol implements an SV-source with parameter $\gamma = \frac{1}{2}(1 + O(\frac{t}{n}))$, we derive Theorem 5.1 as a corollary of Proposition 5.2. Furthermore, Proposition 5.2 yields an alternative way of recognizing BPP languages in polynomial time using an arbitrary SV-source with parameter $\gamma < \frac{1}{\sqrt{2}} \approx 0.7$. Consider, without loss of generality, an algorithm A that using n (perfect) random coins

errs with probability at most ϵ, where $\epsilon > 0$ is a small constant (depending on γ). In order to utilize A when only an SV-source is available, we first use algorithm A_1 (with the SV-source) to generate a "somewhat random" n-bit string, r, and then invoke algorithm A with the string r as a substitute for the n coins required by A. We stress that algorithm A is only invoked once. To analyze the performance of the new algorithm, let S be the set of coin sequences on which A errs. By our hypothesis $|S| \leq \epsilon \cdot 2^n$ and thus using Proposition 5.2 a string $r \in S$ is generated with probability at most $\log(1/\epsilon) \cdot \epsilon^{\log(1/\gamma)} < 1/3$ for sufficiently small $\epsilon > 0$. Thus, using an SV-source (with parameter $\gamma < \frac{1}{\sqrt{2}}$), our algorithm errs with probability at most 1/3.

The logarithmic factors in Theorem 5.1 and Proposition 5.2 can be eliminated; see subsection 5.3.

5.2. Proof of Proposition 5.2. Following is a description of the *algorithm* A_1. The constant δ used in the description will be determined later (as a function of γ). On input 1^n, the algorithm proceeds in rounds, each round consisting of two steps. In the first step, algorithm A_1, uses a second sampling algorithm, denoted A_2, to select a succinct description of a "pseudorandom" partition of the residual sample space. In the second step, algorithm A_1 uses the next bit of the SV-source to determine a side of this partition and so further restricts the residual sample space. We use two types of partitions. In the first $n - 4\log_2 \delta n$ rounds, algorithm A_1 uses partitions defined, as in subsection 4.3, by a polynomial of degree $(\delta n)^4$ over $GF(2^n)$. In the remaining rounds, where the residual sample space is most likely to be smaller than $2(\delta n)^4$, algorithm A_1 uses partitions uniformly chosen from the set of all *perfectly balanced* partitions (i.e., bipartitions in which the cardinalities of the two sides are either equal or differ by one). The two-step process is repeated until the residual sample space contains a unique element. We will see that algorithm $A_1(1^n)$ almost certainly halts after no more than $n + 2$ rounds. (Longer executions can be truncated after $n + 2$ rounds with an arbitrary output.)

We now turn to the description of *algorithm* A_2, which is invoked by $A_1(1^n)$ on input 1^m, where $m = (\delta n)^4 \cdot n$ for the first $n - 4\log_2 \delta n$ rounds of $A_1(1^n)$ and where m is the size of the residual sample space of A_1 later on. On input 1^m, algorithm A_2 proceeds in m rounds. In the ith round, the algorithm uses a third sampling algorithm, denoted A_3, to selects a random m-dimensional binary vector v_i that is linearly independent of previously used vectors. Clearly, the candidate vectors constitute an $(m - (i - 1))$-dimensional vector space over GF_2. The chosen vector partitions the residual sample space into two subsets of equal cardinality (as in Proposition 4.24). Algorithm A_2 uses the next bit of the SV-source to select a side of this partition.

Algorithm A_3, invoked by $A_2(1^m)$, on input 1^k (for $k = m, m-1, \ldots, 1$), is the trivial sampling algorithm which generates a sample point in $\{0,1\}^k$ by merely using the next k bits of the SV-source.

We now turn to the analysis of the sampling algorithm A_1. We first consider what happens if one replaces algorithm A_2 by an algorithm that uniformly selects the appropriate partitions (i.e., $(\delta n)^4$-degree polynomial for the first $n - 4\log_2 \delta n$ rounds and perfectly balanced partitions for later rounds). The analysis is done following the paradigm of the previous section. Namely, we first analyze the performance of the algorithm assuming it employs partitions which satisfy some combinatorial properties (cf., Claim 5.2.1), and next consider the probability that uniformly selected partitions satisfy these properties (cf., Claim 5.2.2).

Claim 5.2.1 (A_1 with balanced partitions). Let U_i be the residual sample space after round i, and $S_i \stackrel{\text{def}}{=} S \cap U_i$ ($U_0 \stackrel{\text{def}}{=} \{0,1\}^n$). Suppose that, for every i, algorithm

A_1 partitions U_{i-1} in a way that is Δ-balanced with respect to S_{i-1} as well as to U_{i-1}. Furthermore, suppose that for every $i > n - 4\log_2 \Delta$, the ith partition chosen for algorithm A_1 is perfectly balanced (i.e., $-1 \leq 2|U_i| - |U_{i-1}| \leq 1$.) Then

$$\text{Prob}(A_1(1^n) \in S) \leq 2\Delta \cdot p^{\log_2(1/\gamma)}.$$

In addition, $|U_{n-4\log_2 \delta n}| < 2(\delta n)^4$, provided that $\Delta \leq \delta n$.

Proof. The proof is analogous to the proof of Corollary 4.8. Using an argument analogous to one used in the proof of Lemma 4.5, we conclude that after $t \stackrel{\text{def}}{=} n - \log_2(1/p)$ rounds the residual sample space contains at most Δ elements of S (i.e., $|S_t| \leq \Delta$). Actually, the argument only uses the hypothesis that the ith partition is Δ-balanced with respect to S_{i-1}, for every $i \leq t$, and is indifferent to the way in which the sides of the partitions are selected in these t rounds. Using the hypothesis that the ith partition is Δ-balanced with respect to U_{i-1}, for every $i \leq t$, we conclude that after these t rounds, the residual sample space contains at least $\frac{1}{2p}$ elements (i.e., $|U_t| \geq 1/2p$). Furthermore, using the hypothesis that also the following $s \stackrel{\text{def}}{=} \log_2(1/p) - 4\log_2 \Delta$ rounds use partitions which are Δ-balanced with respect to the residual sample space, we conclude that after $t + s = n - 4\log_2 \Delta$ rounds the residual sample space has cardinality at least $\frac{1}{2}\Delta^4$ (use Claim 4.4). Now, since all the remaining partitions are assumed to be perfectly balanced, there must be at least $l \stackrel{\text{def}}{=} (4\log_2 \Delta) - 1$ rounds until termination. We now return to the situation after t rounds, and consider the remaining rounds, which by the above are at least $r \stackrel{\text{def}}{=} s + l = \log_2(1/p) - 1$ in number. Since the side of the partition is selected by an SV-source with parameter γ, the probability that any specific element in U_t survives the remaining (i.e., at least r) rounds is at most γ^r. Thus, the probability that some element of S_t survives these rounds does not exceed

$$\begin{aligned} |S_t| \cdot \gamma^r &\leq \Delta \cdot \gamma^{\log_2(1/p)-1} \\ &\leq \Delta \cdot p^{\log_2(1/\gamma)} \cdot 2^{\log_2(1/\gamma)}. \end{aligned}$$

But $\gamma \geq 1/2$, whence $\log_2(1/\gamma) \leq 1$ and the main part of the claim follows.

The additional part (i.e., $|U_{n-4\log_2 \delta n}| < 2(\delta n)^4$) follows easily by using Claim 4.4. □

Claim 5.2.2 (A_1 — probability of balanced partitions). For every $\epsilon > 0$ there exists a $\delta > 0$ so that the following holds. Let π_i denote the probability that a *uniformly chosen* partition for round i is not $\delta \cdot \log_2(1/p)$-balanced with respect to either S_{i-1} or U_{i-1}. Then,

$$\sum_{i \geq 1} \pi_i^{\epsilon} < p.$$

As in Proposition 4.23, it is very useful for the sequel (though, admittedly, not very natural) to raise the probabilities to the ϵth power.

Proof. For $i \leq n - 4\log_2 \delta n$, the proof is identical to the simpler cases (e.g., Properties (P0) and (P1)) considered in the proof of Proposition 4.23. For $i > n - 4\log_2 \delta n$, we observe that the probability of any event, assuming a uniformly selected *perfectly-balanced* partition is at most $\sqrt{|U_{i-1}|}$ times larger than its probability assuming a uniformly selected partition. Since the argument of Proposition 4.23 can tolerate such factors, the claim follows also for $i > n - 4\log_2 \delta n$. □

Combining Claims 5.2.1 and 5.2.2, we conclude that it suffices to show that for some constant $\epsilon > 0$ and for any set of "bad" partitions, $B \subseteq \{0,1\}^m$, the probability that $A_2(1^m)$ produces an output in B is at most $(|B|/2^m)^\epsilon$. Once this is done, the proposition follows by considering $B^{(i)}$, the set of partitions which are not $\delta \cdot \log_2(1/p)$-balanced with respect to either S_{i-1} or U_{i-1}, and noting that $\delta \cdot \log_2(1/p) \leq \delta n$ (which guarantees that in the last $4\log_2(\delta \log_2(1/p))$ rounds perfectly-balanced partitions are used). Namely,

$$\begin{aligned}\text{Prob}(A(1^n) \in S) &< \text{Prob}(A(1^n) \in S | \forall i\ A(1^m) \notin B^{(i)}) \\ &\quad + \text{Prob}(\exists i \text{ s.t. } A(1^m) \in B^{(i)}) \\ &< 2\delta \log(1/p) \cdot p^{\log_2(1/\gamma)} + \sum_{i \geq 1} \left(\frac{|B^{(i)}|}{2^m} \right)^\epsilon \\ &< 3\delta \log(1/p) \cdot p^{\log_2(1/\gamma)},\end{aligned}$$

where the second inequality is based on Claim 5.2.1 and our hypothesis concerning A_2 and the last inequality follows from Claim 5.2.2. Also note that Claim 5.2.1 guarantees that the residual sample space after $n - 4\log_2(\delta n)$ rounds has size at most $\text{poly}(n)$, whence it is possible to represent and generate random partitions of it. Thus, we turn to the analysis of algorithm A_2. Recall that our goal is to show that for some ϵ, (that depends on γ), and for every $B \subseteq \{0,1\}^m$ of cardinality $q \cdot 2^m$,

$$\text{Prob}(A_2(1^m) \in B) = O(q^\epsilon). \tag{15}$$

Let $\epsilon \stackrel{\text{def}}{=} \log_2(1/\gamma) - \frac{1}{2} > 0$ and $\beta \stackrel{\text{def}}{=} \frac{1}{1+\epsilon} < 1$ (recall that $\gamma < \frac{1}{\sqrt{2}}$ is assumed). Also, $\epsilon \leq \frac{1}{2}$ and $\beta \geq \frac{2}{3}$, since $\gamma \geq \frac{1}{2}$. Henceforth, we fix an arbitrary set $B \subseteq \{0,1\}^m$ and let $q \stackrel{\text{def}}{=} \frac{|B|}{2^m}$ (as above). We separately analyze the performance of A_2 throughout the first t rounds (hereafter referred to as *phase* 1), and in the remaining $m - t$ rounds (*phase* 2), where

$$t \stackrel{\text{def}}{=} \max\{0, m - \frac{2\beta}{2\beta - 1} \log_2(1/q)\}. \tag{16}$$

Let B_i denote the residual set (of bad polynomials) after i rounds of algorithm A_2 (e.g., $B_0 = B$).

Claim 5.2.3 (A_2 — *phase* 1).

$$\text{Prob}(|B_t| > 2q \cdot 2^{m-t}) \leq O(q^{2\epsilon}).$$

That is, the probability that B_t is greater than twice its "expected size" is small. Note that by definition of t, we have $m - t = \frac{2\beta}{2\beta-1} \cdot \log_2(1/q)$ and $q \cdot 2^{m-t} = 2^{(m-t)/2\beta}$.

Proof. For every i, let $b_i \stackrel{\text{def}}{=} \frac{|B|}{2^i}$. Our plan is to prove that with very high probability, $|B_i| \approx b_i$ for every $i \leq t$, which would establish our claim. We consider the first time when $|B_i| \not\approx b_i$. Thus, the probability that $|B_t| > 2q \cdot 2^{m-t}$ is bounded above by

$$\text{Prob}\left[\exists i < t : \left(\left| |B_{i+1}| - \frac{|B_i|}{2} \right| > |B_i|^{\frac{1+\beta}{2}} \right) \wedge \left(\forall j < i : \left| |B_{j+1}| - \frac{|B_j|}{2} \right| \leq |B_j|^{\frac{1+\beta}{2}} \right) \right].$$

Now, using Chebyshev's Inequality (as in the proof of Proposition 4.24), we can show that *for a uniformly chosen random linear partition*,

$$\text{Prob}\left(\left||B_{i+1}| - \frac{|B_i|}{2}\right| > |B_i|^{\frac{1+\beta}{2}}\right) < \frac{1}{|B_i|^{\beta}}.$$

Call a linear partition for round $i+1$ *bad*, if $\left||B_{i+1}| - \frac{|B_i|}{2}\right| > |B_i|^{\frac{1+\beta}{2}}$. We now know that the number of bad partitions is bounded by $\frac{1}{|B_i|^{\beta}} \cdot 2^{m-i}$. We need to bound the probability that $A_3(1^{m-i})$ selects a bad partition in round $i+1$. Using the union bound, the definition of A_3 and Claim 4.4 (for $|B_i|$), we have

$$\begin{aligned}\text{Prob}(A_3(1^{m-i}) \text{ is bad}) &\leq \frac{2^{m-i}}{|B_i|^{\beta}} \cdot \gamma^{m-i} \\ &< 2 \cdot \gamma^{m-i} \cdot \frac{2^{m-i}}{b_i^{\beta}},\end{aligned}$$

where the last inequality uses our assumption that all previous partitions are good (whence for each $j < i$, $|B_{j+1}| > \frac{|B_j|}{2} - |B_j|^{\frac{1+\beta}{2}}$ and, consequently, $|B_i| > \frac{b_i}{2}$). Since $b_i = 2^{t-i} \cdot b_t$ and $b_t^{\beta} = (q2^{m-t})^{\beta} = 2^{(m-t)/2}$ (see remark above), we get

$$\begin{aligned}\text{Prob}(A_3(1^{m-i}) \text{ is bad}) &< 2 \cdot \gamma^{m-i} \cdot \frac{2^{m-i}}{2^{(t-i)\beta} \cdot 2^{(m-t)/2}} \\ &= 2 \cdot \gamma^{m-i} \cdot 2^{(m-i)-\frac{m-t}{2}-\beta(t-i)} \\ &= 2 \cdot \left(\gamma\sqrt{2}\right)^{m-i} \cdot 2^{-(\beta-\frac{1}{2})\cdot(t-i)}.\end{aligned}$$

Letting $\rho \stackrel{\text{def}}{=} \gamma \cdot \sqrt{2} < 1$ (as $\gamma < \frac{1}{\sqrt{2}}$) and using $m - i \geq m - t > 2\log_2(1/q)$ (as $m - t = \frac{2\beta}{2\beta-1}\log_2(1/q)$ and $\beta < 1$), we get

$$\begin{aligned}\text{Prob}(A_3(1^{m-i}) \text{ is bad}) &< 2 \cdot \rho^{2\log_2(1/q)} \cdot 2^{-(\beta-\frac{1}{2})\cdot(t-i)} \\ &= 2 \cdot q^{2\log_2(1/\rho)} \cdot 2^{-(\beta-\frac{1}{2})\cdot(t-i)} \\ &= a \cdot b^{t-i},\end{aligned}$$

where $a \stackrel{\text{def}}{=} 2q^{2\log_2(1/\rho)}$ and $b \stackrel{\text{def}}{=} 2^{-(\beta-\frac{1}{2})} < 1$ (as $\beta > \frac{1}{2}$). Hence, the probability that A_3 chooses a bad partition for some round i, throughout phase 1, is bounded by $\sum_{i=1}^{t} a \cdot b^{t-i} < \frac{a}{1-b}$. Using $\epsilon = \log_2(1/\gamma) - \frac{1}{2} = \log_2(1/\rho) \leq \frac{1}{2}$ and $\beta = \frac{1}{1+\epsilon}$, we get

$$\begin{aligned}\frac{a}{1-b} &= \frac{2q^{2\epsilon}}{1 - 2^{-\frac{1-\epsilon}{2(1+\epsilon)}}} \\ &\leq \frac{2q^{2\epsilon}}{1 - 2^{-1/6}} \\ &< 20 \cdot q^{2\epsilon},\end{aligned}$$

and the claim follows. □

Claim 5.2.4 (A_2 — *phase* 2). Let B_t be the residual target set after t rounds and consider an execution of the $m - t$ remaining rounds. Suppose that $|B_t| \leq 2b_t$, where

$b_t \stackrel{\text{def}}{=} \frac{|B|}{2^t}$ (as in Claim 5.2.3). Then the probability that $A_2(1^m)$ terminates with output in B_t is at most $2q^\epsilon$.

Proof. We consider the executions of rounds $t+1$ through m. Regardless of which linear partitions are used in the remaining $m-t$ rounds, the probability that a particular element of B_t is output by $A_2(1^m)$ is bounded by γ^{m-t}. Hence,

$$\begin{aligned}\text{Prob}(A_2(1^m) \text{ hits } B_t) &\le |B_t| \cdot \gamma^{m-t} \\ &\le 2b_t \cdot \gamma^{m-t} \\ &= 2 \cdot 2^{\frac{m-t}{2\beta}} \cdot \gamma^{m-t} \\ &= 2 \cdot \left(\gamma \cdot 2^{\frac{1}{2\beta}}\right)^{m-t}.\end{aligned}$$

Setting (as before) $\rho = \gamma\sqrt{2}$, and using $\epsilon = \log_2(1/\rho)$ and $\beta = \frac{1}{1+\epsilon}$, we get $2^{\frac{1}{2\beta}} = \sqrt{2/\rho}$. Hence, using again $\rho < 1$ and $m - t > 2\log_2(1/q)$, we get

$$\begin{aligned}\text{Prob}(A_2(1^m) \text{ hits } B_t) &\le 2 \cdot \left(\gamma \cdot \sqrt{\frac{2}{\rho}}\right)^{m-t} \\ &< 2 \cdot \sqrt{\rho}^{\,2\log_2(1/q)} \\ &= 2q^{\log_2(1/\rho)} \\ &= 2q^\epsilon,\end{aligned}$$

and the claim follows. □

Combining Claims 5.2.3 and 5.2.4, we have established equation (15) and the proposition follows. □

5.3. Further improvements. Actually, the result of Proposition 5.2 can be improved using a slightly more careful analysis of the algorithm A_1 provided in the above proof. The improved analysis is analogous to the proof of the tighter bound for the case $q_v = p_v$ of Theorem 4.27. Namely, we replace Claims 5.2.1 and 5.2.2 by the following three claims. In the first two claims we assume that algorithm A_2 satisfies equation (15).

Claim 1. With probability at least $1-p$, after $i \stackrel{\text{def}}{=} n - \log_2(1/p) - 4\log_2(\delta\log_2(1/p))$ rounds the residual sample space contains at most $2(\delta\log_2(1/p))^4$ elements of S; namely,

$$\text{Prob}(|S_i| > 2(\delta\log_2(1/p))^4) < p.$$

Claim 2. Consider an arbitrary subset S' of U_t so that $|S'| \le 2(\delta\log_2(1/p))^4$. Then the expected number of elements of S' which survive an additional number of $4\log_2(\delta\log_2(1/p))$ rounds is bounded above by $O(1)$.

Claim 3. Let $t \stackrel{\text{def}}{=} n - \log_2(1/p)$. Then,

$$\text{Prob}(A(1^n) \in S) \le \text{Exp}(|S_t|) \cdot \gamma^{\log_2(1/p)-1}.$$

(Here, we do use a part of the proof of Claim 5.2.1 to assert that with probability $1-p$ the protocol does not terminate before $n-1$ rounds.)

Consequently, we get

- Improvement to Proposition 5.2: *For every constant* γ, $\frac{1}{2} \leq \gamma < \frac{1}{\sqrt{2}}$, *the algorithm* A_1 *appearing in the proof of Proposition* 5.2 *satisfies, for every set* $S \subseteq \{0,1\}^n$,

$$\mathrm{Prob}(A_1(1^n) \in S) = O(p^{\log_2(1/\gamma)}),$$

  where $p \stackrel{\mathrm{def}}{=} |S|/2^n$, *and the probability is taken over an arbitrary SV-source with parameter* γ.
- Theorem 5.1 can be improved analogously. Namely, *for every set* $S \subseteq \{0,1\}^l$, *if* $m - t$ *parties plays honestly then the outcome of the protocol is in* S *with probability bounded above by* $O(p^{1-O(\frac{t}{m})})$, *where* $p \stackrel{\mathrm{def}}{=} |S|/2^l$.

Acknowledgments. Oded Goldreich would like to thank the Computer Science Department of the Hebrew University for providing him shelter in times of war. The authors wish to thank the Computer Science Department of Tel-Aviv University for use of its computing facilities.

REFERENCES

[1] N. Alon and M. Naor, *Coin-flipping games immune against linear-sized coalitions*, SIAM J. Comput., 22 (1993), pp. 403–417.

[2] D. Beaver and S. Goldwasser, *Distributed computation with faulty majority*, in Proc. 30th Annual IEEE Symp. on Foundations of Computing, IEEE Computer Society Press, Los Alamitos, CA, 1989, pp. 468–473.

[3] M. Ben-or, S. Goldwasser, and A. Wigderson, *Completeness theorems for non-cryptographic fault-tolerant distributed computation*, in Proc. 20th Annual ACM Symp. on Theory of Computing, ACM, New York, 1988, pp. 1–10.

[4] M. Ben-or and N. Linial, *Collective coin flipping*, in Randomness and Computation, S. Micali, ed., JAI Press, Greenwich, CT, 1989, pp. 91–115.

[5] M. Ben-or, N. Linial, and M. Saks, *Collective coin flipping and other models of imperfect randomness*, Colloq. Math Soc. János Bolyai 52, Combinatorics Eger 1987, pp. 75–112.

[6] B. Chor and O. Goldreich, *Unbiased bits from sources of weak randomness and probabilistic communication complexity*, SIAM J. Comput., 17 (1988), pp. 230–261.

[7] D. Chaum, C. Crepeau, and I. Damgård, *Multiparty unconditionally secure protocols*, in Proc. of 20th Annual ACM Symp. on Theory of Computing, ACM, New York, 1988, pp. 11–19.

[8] B. Chor and E. Kushilevitz, *A zero-one law for boolean privacy*, SIAM J. Discrete Math., 4 (1991), pp. 36–47.

[9] J. Cooper and N. Linial, *Fast perfect-information leader election protocol with linear immunity*, in 25th Annual ACM Symposium on the Theory of Computing, San Diego, ACM, New York, 1993, pp. 662–671; Combinatorica, to appear.

[10] I. Damgård, *Interactive hashing can simplify zero-knowledge protocol design without computational assumptions*, in Advances in Cryptology, Proceedings of Crypto93, Lecture Notes in Computer Science 773, Springer-Verlag, New York, 1983, pp. 100–109.

[11] Z. Galil, S. Haber, and M. Yung, *Cryptographic computation: Secure faulty-tolerant protocols and the public key model*, in Advances in Cryptology, Proceedings of Crypto87, Lecture Notes in Computer Science 293, Springer-Verlag, New York, 1987, pp. 135–155.

[12] O. Goldreich, S. Goldwasser, and N. Linial, *Fault-tolerant Computation in the Full Information Model*, Tech. report TR-682, Computer Science Dept., Technion, Haifa, Israel, July 1991.

[13] O. Goldreich, S. Micali, and A. Wigderson, *Proofs that yield nothing but their validity and a methodology for cryptographic protocol design*, J. Assoc. Comput. Mach., 38 (1991), pp. 691–729.

[14] O. Goldreich, S. Micali, and A. Wigderson, *How to play any mental game*, in Proc. 19th Annual ACM Symp. on Theory of Computing, ACM, New York, 1987, pp. 218–229.

[15] S. Goldwasser and L. A. Levin, *Fair computation of general functions in presence of immoral majority*, in Advances in Cryptology, Proceedings of Crypto90, Lecture Notes in Computer Science 537, Springer-Verlag, New York, 1990, pp. 77–93.

[16] J. Kahn, G. Kalai, and N. Linial, *The influence of variables on boolean functions*, in 29th Annual IEEE Symp. on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 68–80.

[17] J. Kilian, *Founding cryptography on oblivious transfer*, in Proc. of 20th Annual ACM Symp. on Theory of Computing, ACM, New York, pp. 20–29, 1988.

[18] M. Naor, R. Ostrovsky, R. Venkatesan, and M. Yung, *Perfect zero-knowledge arguments for NP can be based on general complexity assumptions*, in Advances in Cryptology, Proceedings of Crypto92, Lecture Notes in Computer Science 740, Springer-Verlag, New York, 1992; *J. Cryptology*, to appear.

[19] R. Ostrovsky, S. Rajagopalan, and U. Vazirani, *Simple and efficient leader election in the full information model*, in Proc. 26th Annual ACM Symp. on Theory of Computing, ACM, New York, 1994, pp. 234–242.

[20] R. Ostrovsky, R. Venkatesan, and M. Yung, *Fair games against an all-powerful adversary*, in DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 13, Jin-Yi Cai ed., AMS, Providence, RI, 1993, pp. 155–169.

[21] R. Ostrovsky, R. Venkatesan, and M. Yung, *Interactive hashing simplifies zero-knowledge protocol design*, in Proc. Eurocrypt93, Lecture Notes in Computer Science 765, Springer-Verlag, New York, 1983, pp. 267–273.

[22] M. Santha and U. V. Vazirani, *Generating quasi-random sequences from slightly-random sources*, in 25th Annual IEEE Symp. on Foundation of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1984, pp. 434–440.

[23] U. V. Vazirani and V. V. Vazirani, *Random polynomial time is equal to slightly-random polynomial time*, in 26th Annual IEEE Symp. on Foundation of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1985, pp. 417–428.

[24] A. C. Yao, *Protocols for secure computations*, in 23rd Annual IEEE Symp. on Foundations of Computer Science, 1982, pp. 160–164.

[25] A. C. Yao, *How to generate and exchange secrets*, in 27th Annual IEEE Symp. on Foundations of Computer Science, 1986, pp. 162–167.

SIAM J. COMPUT.
Vol. 27, No. 2, pp. 545–556, April 1998
© 1998 Society for Industrial and Applied Mathematics
011

A SPECTRAL APPROACH TO LOWER BOUNDS WITH APPLICATIONS TO GEOMETRIC SEARCHING*

BERNARD CHAZELLE†

Abstract. We establish a nonlinear lower bound for halfplane range searching over a group. Specifically, we show that summing up the weights of n (weighted) points within n halfplanes requires $\Omega(n \log n)$ additions and subtractions. This is the first nontrivial lower bound for range searching over a group. By contrast, range searching over a semigroup (which forbids subtractions) is almost completely understood.

Our proof has two parts. First, we develop a general, entropy-based method for relating the linear circuit complexity of a linear map A to the spectrum of $A^\top A$. In the second part of the proof, we design a "high-spectrum" geometric set system for halfplane range searching and, using techniques from discrepancy theory, we estimate the median eigenvalue of its associated map. Interestingly, the method also shows that using up to a linear number of help gates cannot help; these are gates that can compute *any* bivariate function.

Key words. lower bounds, eigenvalues, range searching, circuit complexity

AMS subject classifications. 68P05, 68Q20, 68R99, 51M99

PII. S0097539794275665

1. Introduction.

Given n weighted points in the plane and n halfplanes, we consider the classical *halfplane range searching* problem, which is to compute the sum of the weights of the points within each of the given regions. If subtractions are not allowed (the semigroup model) the problem is almost completely solved [7, 11, 15]; see also [9, 16, 17, 19] for surveys of the vast literature on the subject. In the (commutative) group model, where subtractions are allowed, there is little evidence that any power should be gained beyond polylog speedups, but proving it has been elusive. In fact, in that model no superlinear lower bound has ever been established for any range searching problem of any kind. The problem is equivalent to asking for the nonmonotone circuit complexity of some fairly unwieldy linear transformation over the reals, so the lack of progress should not come as a big surprise.

This paper takes a first, modest step toward resolving this question. We establish a lower bound of $\Omega(n \log n)$ on the complexity of range searching with respect to n points and n halfplanes (given in advance). The model of computation is a straight-line program: each step performs a group operation of the form

$$z \leftarrow x \pm y,$$

where x and y are previously computed variables or input weights. The underlying group is assumed to be commutative. Note that it is easy to prove an $\Omega(n \log n)$ lower bound by reduction from sorting, but this says nothing about the number of times weights have to be added or subtracted. In the group model, memory accesses are not

*Received by the editors October 17, 1994; accepted for publication (in revised form) March 20, 1996. A preliminary version of this paper appeared as *A spectral approach to lower bounds*, in Proceedings of the 35th Annual IEEE Symposium on the Foundations of Computer Science (FOCS), IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 674–682.

http://www.siam.org/journals/sicomp/27-2/27566.html

†Department of Computer Science, Princeton University, Princeton, NJ 08544 (chazelle@cs.princeton.edu). This research was supported in part by NSF grant CCR-93-01254 and by The Geometry Center, University of Minnesota, a Science and Technology Center funded by the NSF, DOE, and Minnesota Technology, Inc.

charged; only group operations are. Note that this makes lower bounds even stronger. The program must work for all groups and all weight assignments.

This formulation of the group model is probably the most natural. But one might ask what happens if we extend the model by allowing extra computations free of charge. For example, can adding the same variable a large number of times, e.g., $z \leftarrow Mx$ (M integral), possibly help? How about encoding special functions in lookup tables? In general, we call a *free computation* any assignment of the form

$$z \leftarrow f(x, y),$$

where f is an arbitrary function. We show that allowing close to $n/4$ free computations cannot help. To put this result in perspective, one should note that over the reals $2n$ free computations suffice to make the problem trivial.

THEOREM 1.1. *Range searching with respect to n points and n halfplanes requires $\Omega(n \log n)$ group operations. This remains true even with the help of up to $n/4 - \varepsilon n$ free computations, for any fixed $\varepsilon > 0$. On the other hand, over the reals, the problem can be solved in linear time with only $2n - 1$ free computations.*

It is likely that the lower bound is far from optimal. The best known upper bound is slightly above $O(n^{4/3})$ [15], and in the semigroup model the best lower bound (for the on-line version) is also $\Omega(n^{4/3})$ [7]. On the brighter side, Theorem 1.1 provides the only lower bound known for the group model, so at least it is a step in the right direction.

Proof. The proof consists of two distinct parts. First, we establish a general *spectral lemma*, which asserts a lower bound of $\Omega((k-2m)\log \lambda_k)$ on the linear circuit complexity of any linear transformation A (with integer coefficients) from $\mathbf{R}^n$ to $\mathbf{R}^n$, where λ_k is the kth largest eigenvalue of $A^\top A$, and m is the number of help gates. These are the circuit equivalent of free computations. (This shows that allowing up to roughly $k/2$ help gates does no good.) The lower bound holds for any value of k between $2m$ and n. This freedom is useful because often only a small range of the whole spectrum can be accurately estimated without too much effort.

In the second part of the proof, we design a *hard* instance of range searching by nonconstructive means. Then we use spectral methods from discrepancy theory to estimate the median eigenvalue of the quadratic form associated with the corresponding set system.

Remark 1. Our technique trivially implies an $\Omega(n \log n)$ lower bound for range searching over a finite projective plane. In general, the technique will yield a lower bound on any instance of range searching whose corresponding spectrum can be mapped out reasonably well. Powerful techniques in discrepancy theory [4], such as those used in section 3 of this paper, raise hope that more lower bounds can be derived by this approach.

Remark 2. A simple application of the spectral lemma is that computing Hx, where H is the $n \times n$ Sylvester Boolean matrix, takes $\Omega(n \log n)$ time even in the presence of about $n/2$ help gates. (All the eigenvalues λ_k of $H^\top H$ are equal to n.) Note that the choice of ground field is crucial, since Hx can be computed in linear time over GF(2) [2]. If we forbid help gates, the same bound can be obtained more simply by using Morgenstern's volume argument [18]. The spectral lemma works in a model that is ideally suited for range searching. If, instead of a group, the linear transformation operates over a ring or a field (like the discrete Fourier transform), then for the lemma to hold, the (nonhelp) gates must evaluate linear forms with bounded coefficients. This is the same limitation found in Morgenstern's result. Help gates can be thought of as a way of partly overcoming this limitation.

Remark 3. The proof of the spectral lemma is based on entropy considerations. By avoiding standard volume arguments, we are able to accommodate help gates. Indeed, a weakness of the volume argument of [18] is that it collapses even in the presence of a single help gate. Intuitively, the idea of that argument is to relate the work of the circuit to the volume of the ellipsoid into which the circuit "transforms" the unit sphere. Any such argument is vulnerable to even a single help gate, because any one of them has the ability to blow up the entire sphere. On the contrary, our entropy-based approach ensures that the "contribution" of a single help gate to the work of the circuit is always bounded, regardless of the gate's power.

There has been a substantial amount of work in arithmetic circuit complexity—see surveys in [13, 20]—but, to our knowledge, nothing that allows us to tackle a geometric problem such as range searching. Most of the recent research in circuit complexity [5], including work involving help bits or oracle queries (variants of our help gates) [3, 6], has been mostly concerned with problems over finite fields and seems of little relevance to our problem.

2. Eigenvalues, entropy, and linear circuits. Let A be an $n \times n$ matrix with integer elements. A linear circuit for computing $y = Ax$, where $x \in \mathbf{R}^n$, is a directed acyclic graph with n input nodes $x = (x_1, \ldots, x_n)$ and n output nodes $y = (y_1, \ldots, y_n)$. The size of the circuit is its number of edges. A node is a gate that computes a real-valued function

$$f(z_1, z_2) = \alpha_1 z_1 + \alpha_2 z_2,$$

where $z_i \in \mathbf{R}$ and $\alpha_i \in \{-1, 0, 1\}$. In addition, we allow the presence of m help gates: these gates can evaluate *any* function $f(z_1, z_2)$ from $\mathbf{R}^2$ to $\mathbf{R}$. Recall that the matrix $M \stackrel{\text{def}}{=} A^\top A$ is diagonalizable and its eigenvalues are real:

$$\lambda_1 \geq \cdots \geq \lambda_n \geq 0.$$

All logarithms are to the base 2.

SPECTRAL LEMMA. *Given any $1 \leq k \leq n$, any circuit for computing Ax has size at least $c(k - 2m) \log \lambda_k$, for some constant $c > 0$, where $m \leq k/2$ is the number of help gates and λ_k is the kth largest eigenvalue of $A^\top A$.*

Proof. Let K be the invariant subspace spanned by the eigenvectors for M associated with $\lambda_1, \ldots, \lambda_k$. We ensure that K is of dimension exactly k by dropping some of the eigenvectors for λ_k, in case of multiplicity. Let $B_n(p, r)$ denote the Euclidean n-ball of radius r centered at p and let $V_n(r)$ be its volume. Consider the cubes of the form $\mathbf{Z}^n + [0, 1]^n$ that intersect $K \cap B_n(O, R)$, for some (large enough) parameter R. Let L be the set of centers of these cubes. Finally, let $H(x)$ denote the entropy of a random variable x with values distributed uniformly in L. We estimate the entropy of x (Lemma 2.1) and then we prove the key lemma, which says that if the spectrum of M is not too low we can suitably hash the image of x under A without losing much entropy (Lemma 2.2).

LEMMA 2.1.

$$H(x) \geq \log V_k(R) - \log V_k(\sqrt{n}\,).$$

Proof. Let $M_R^-(n, k)$ denote the minimum number, over all k-flats F containing the origin, of the n-cubes of the form $\mathbf{Z}^n + [0, 1]^n$ that intersect $F \cap B_n(O, R)$. Observe that the cubes to be counted cover the ball $B_k(O, R)$ embedded in F. Furthermore,

the intersection of F with each of these cubes fits into a k-ball of radius $\sqrt{n}/2$, so

$$M_R^-(n,k) \geq V_k(R)/V_k(\sqrt{n}\,).$$

The lemma follows from the fact that $|L| \geq M_R^-(n,k)$ and $H(x) = \log|L|$. □

Another useful quantity, denoted by $M_r^+(n,m)$, is the maximum number over all m-flats F of the n-cubes of the form $\mathbf{Z}^n + [0,1]^n$ that intersect $F \cap B_n(O,r)$. In the case $n-m=0$, the cubes counted by $M_r^+(n,m)$ all lie within $B_n(O, r+\sqrt{n}\,)$; therefore

$$M_r^+(n,n) \leq V_n(r+\sqrt{n}\,). \tag{1}$$

Assume now that $n-m>0$. In the appendix we show that, for n and r large enough,

$$M_r^+(n,m) \leq 3^n V_m(r). \tag{2}$$

LEMMA 2.2. *If A is a matrix with real elements, such that $1 \leq \lambda_k \leq 2$, then*[1]

$$H(\lfloor Ax \rfloor) \geq H(x) - \log V_n(5\sqrt{n}\,).$$

Proof. Let $x, x' \in L$ be such that $\lfloor Ax \rfloor = \lfloor Ax' \rfloor$. It follows that $\|A(x-x')\|_2 \leq \sqrt{n}$. Write x as the direct sum $x_0 + u$, where $x_0 \in K$, $u \in K^\perp$, and do the same with x'. Observe that

$$\|u-u'\|_2 \leq \|u\|_2 + \|u'\|_2 \leq \sqrt{n}.$$

By the variational characterization of eigenvalues, $\|A(x_0-x_0')\|_2 \geq \sqrt{\lambda_k}\,\|x_0-x_0'\|_2$ and, because $K^\perp$ contains $u-u'$ and is spanned by eigenvectors corresponding to $\lambda_j \leq \lambda_k$, we have $\|A(u-u')\|_2 \leq \sqrt{\lambda_k}\,\|u-u'\|_2$. It follows that (for $1 \leq \lambda_k \leq 2$)

$$\begin{aligned}
\|x-x'\|_2 &\leq \|x_0-x_0'\|_2 + \|u-u'\|_2 \\
&\leq \|A(x_0-x_0')\|_2 + \|u-u'\|_2 \\
&\leq \|A(x-x')\|_2 + \|A(u-u')\|_2 + \|u-u'\|_2 \\
&\leq \sqrt{n} + 3\|u-u'\|_2 \leq 4\sqrt{n}.
\end{aligned}$$

Thus, the preimage of a fixed $z \in \mathbf{R}^n$ under $x \in L \mapsto \lfloor Ax \rfloor$ lies entirely in a ball $B_n(x, 4\sqrt{n}\,)$, where $x \in L$, and therefore, the uniform distribution within that preimage has entropy at most $\log M^+_{4\sqrt{n}}(n,n)$. By (1) this does not exceed $\log V_n(5\sqrt{n}\,)$. Standard identities on the entropy of joint distributions, namely,

$$H(x) = H(x, \lfloor Ax \rfloor) = H(\lfloor Ax \rfloor) + H(\,x \mid \lfloor Ax \rfloor\,),$$

complete the proof. □

Let $z = (z_1, \ldots, z_s)$ be the vector of $\mathbf{R}^n$ whose coordinates are the intermediate variables computed by the gates. For convenience, we append the input variables at the beginning of the list ($z_j = x_j$, for $1 \leq j \leq n$) and the output variables at the end ($z_{s-n+j} = y_j$, for $1 \leq j \leq n$). We also assume that the list corresponds to a topological ordering of the DAG (directed acyclic graph), meaning that for any $j > n$,

$$z_j = \alpha_j z_{f(j)} + \beta_j z_{g(j)},$$

[1] Given $z = (z_1, \ldots, z_n) \in \mathbf{R}^n$, we use the shorthand $\lfloor z \rfloor$ for $(\lfloor z_1 \rfloor, \ldots, \lfloor z_n \rfloor)$.

where $f(j) \le g(j) < j$ and $|\alpha_j|, |\beta_j| \le 1$. In the case of a help gate, z_j is an arbitrary real function of $z_{f(j)}$ and $z_{g(j)}$. Let

$$\mu_k = \lfloor \sqrt{\lambda_k} \rfloor.$$

The input x to the circuit is chosen so that $\tilde{x} = \mu_k x$ is a random variable uniformly distributed in L. We shall now assume that λ_k is large enough, which the spectral lemma obviously allows us to do. We now argue that $\lfloor z \rfloor$ has high entropy. Lemmas 2.4–2.6 will then show that only a large circuit can produce a "hashed" vector $\lfloor z \rfloor$ with that much entropy.

LEMMA 2.3.

$$H(\lfloor z \rfloor) \ge \log V_k(R) - \log V_k(\sqrt{n}\,) - \log V_n(5\sqrt{n}\,).$$

Proof. Because A is a linear map, the circuit outputs $B\tilde{x}$, where $B \stackrel{\text{def}}{=} A/\mu_k$, obviously satisfies the conditions of Lemma 2.2. Thus,

$$H(\lfloor z \rfloor) \ge H(\lfloor B\tilde{x} \rfloor) \ge H(\tilde{x}) - \log V_n(5\sqrt{n}\,).$$

The proof now follows from Lemma 2.1. □

To be able to isolate the action of help gates, we devise the following artifice. Regard the outputs of the help gates, $z_{h(1)}, \ldots, z_{h(m)}$, as new (help) variables, and express each z_j $(1 \le j \le s)$ as a linear form over the set of variables $Z = \{\, x_1, \ldots, x_n, z_{h(1)}, \ldots, z_{h(m)} \,\}$. How this is done is best seen by induction. The input gates are linear forms over single variables. For any other gate z_j, if it is of the helping type, then the form is z_j itself. Otherwise, $z_j = \alpha_j z_{f(j)} + \beta_j z_{g(j)}$ and, by induction, z_j is a linear combination of two linear forms over Z and hence a linear form itself. (Note that even though the outputs of the help gates might be algebraically related, the $z_{h(j)}$ are added by adjunction and so are considered independent.) In the expression for z_j, let z_j^x (resp., z_j^y) denote the linear form obtained by taking only the nonhelp (resp., help) variables. A key step now is to look at z_j^x and z_j^y no longer as linear forms but as real functions $z_j^x(x_1, \ldots, x_n)$ and $z_j^y(x_1, \ldots, x_n)$. While the circuit computes $z_j = z_j^x + z_j^y$, we wish to monitor the information contents of the complex number,[2]

$$z_j^c \stackrel{\text{def}}{=} \lfloor z_j^x \rfloor + i z_j^y.$$

We denote by z^c the vector $(z_1^c, \ldots, z_s^c)$. Our strategy is this: first, we establish that a small circuit can produce only a small entropy $H(z^c)$. We do this in three steps: Lemma 2.4 looks at the effect of the hashing on the entropy of the input variables. Lemmas 2.5 and 2.6 bound how much entropy the nonhelp and help gates, respectively, can inject into the vector of hashed variables. Finally, we show that $H(z^c)$ cannot be much smaller than $H(\lfloor z \rfloor)$, which by Lemma 2.3 is already known to be big.

LEMMA 2.4.

$$H(\, z_1^c, \ldots, z_n^c \,) \le 2n + \log V_k(R/\mu_k).$$

Proof. Let $\mathcal{C}$ be the set of cubes of the form $(\mu_k \mathbf{Z})^n + [0, \mu_k]^n$. Any cube of $\mathcal{C}$ that contains a point of L contains the entire unit cube centered at that point, and so

[2] We use complex numbers simply as a device for representing ordered pairs.

it intersects $K \cap B_n(O,R)$. Thus, the number of such cubes is at most $M^+_{R/\mu_k}(n,k)$, which, by (2), does not exceed $3^n V_k(R/\mu_k)$ so that (recall that $z^c_i = \lfloor x_i \rfloor$, for $i = 1,\ldots,n$)

$$\begin{aligned} H(\, z^c_1,\ldots,z^c_n\,) &= H(\, \lfloor \tilde{x}_1/\mu_k \rfloor,\ldots,\lfloor \tilde{x}_n/\mu_k \rfloor\,) \\ &\le 2n + \log V_k(R/\mu_k). \qquad \square \end{aligned}$$

LEMMA 2.5. *For any nonhelp variable z_j, $n < j \le s$, we have*

$$H(\, z^c_j \,|\, z^c_{f(j)}, z^c_{g(j)}\,) \le 3.$$

Proof. Recall that $z^c_j = \lfloor z^x_j \rfloor + i z^y_j$. Obviously, since the imaginary part is completely determined by those of $z^c_{f(j)}$ and $z^c_{g(j)}$, we have

$$H(\, z^y_j \,|\, z^y_{f(j)}, z^y_{g(j)}\,) = 0.$$

To deal with the real part, we use the inequality $H(\,A\,|\,B\,) \le H(\,A\,|\,C\,) + H(\,C\,|\,B\,)$ to derive

$$\begin{aligned} &H(\, \lfloor z^x_j \rfloor \,|\, \lfloor z^x_{f(j)} \rfloor, \lfloor z^x_{g(j)} \rfloor\,) \\ &\qquad \le H(\, \lfloor z^x_j \rfloor \,|\, \lfloor \alpha_j z^x_{f(j)} \rfloor, \lfloor \beta_j z^x_{g(j)} \rfloor\,) \\ &\qquad\quad + H(\, \lfloor \alpha_j z^x_{f(j)} \rfloor \,|\, \lfloor z^x_{f(j)} \rfloor\,) \\ &\qquad\quad + H(\, \lfloor \beta_j z^x_{g(j)} \rfloor \,|\, \lfloor z^x_{g(j)} \rfloor\,). \end{aligned}$$

Given two random variables ξ, ξ' arbitrarily distributed in $\mathbf{R}$,

$$H(\, \lfloor \xi + \xi' \rfloor \,|\, \lfloor \xi \rfloor, \lfloor \xi' \rfloor\,) \le 1.$$

Intuitively, the only information missing is a one-bit carry. Similarly, given a fixed $\alpha \in \mathbf{Z}$, $|\alpha| \le 1$, and a real random variable ξ, we have $H(\, \lfloor \alpha\xi \rfloor \,|\, \lfloor \xi \rfloor\,) \le 1$. The lemma follows from the fact that $z_j = \alpha_j z_{f(j)} + \beta_j z_{g(j)}$. $\square$

LEMMA 2.6. *For any help variable z_j, $n < j \le s$, we have $H(\, z^c_j \,|\, z^c_{f(j)}, z^c_{g(j)}\,) \le 2(\log \mu_k + 1)$.*

Proof. We have $z^c_j = i z^y_j$, where z^y_j is an arbitrary function of $z_{f(j)}$ and $z_{g(j)}$. Regarding $z_{f(j)} = z^x_{f(j)} + z^y_{f(j)}$, the only information we have at our disposal is $z^c_{f(j)} = \lfloor z^x_{f(j)} \rfloor + i z^y_{f(j)}$. There is no loss of information in the imaginary part. The same is not true of the real part, however. The key observation is that $z^x_{f(j)}$ is a linear form over $x_1,\ldots,x_n$ with integer coefficients. Thus, since $2\mu_k x_i$ is itself integral, so is $2\mu_k z^x_{f(j)}$. It follows that the fractional part of $z^x_{f(j)}$ can be one of only $2\mu_k$ possible values, and hence, $H(\, z^x_{f(j)} \,|\, \lfloor z^x_{f(j)} \rfloor) \le \log \mu_k + 1$, from which the lemma follows. $\square$

We can use the last three lemmas to upper bound $H(\lfloor z \rfloor)$:

$$\begin{aligned} H(z^c) &= H(\, z^c_1,\ldots,z^c_n\,) + \sum_{n+1 \le j \le s} H(\, z^c_j \,|\, z^c_1,\ldots,z^c_{j-1}\,) \\ &\le H(\, z^c_1,\ldots,z^c_n\,) + \sum_{n+1 \le j \le s} H(\, z^c_j \,|\, z^c_{f(j)}, z^c_{g(j)}\,). \end{aligned}$$

By Lemmas 2.4, 2.5, and 2.6,

$$\begin{aligned} H(z^c) &\le 2n + \log V_k(R/\mu_k) + 3(s-n-m) + 2m(\log \mu_k + 1) \\ &\le 3s - n + \log V_k(R/\mu_k) + 2m \log \mu_k, \end{aligned}$$

and therefore,

$$\begin{aligned} H(\lfloor z \rfloor) &\le H(z^c, \lfloor z \rfloor) = H(z^c) + H(\lfloor z \rfloor \mid z^c) \\ &\le H(z^c) + \sum_{j=n+1}^{s} H(\lfloor z_j^x + z_j^y \rfloor \mid \lfloor z_j^x \rfloor + i z_j^y) \\ &\le 4s + 2m \log \mu_k + \log V_k(R/\mu_k). \end{aligned}$$

Bringing the lower bound of Lemma 2.3 to bear, we derive

$$\begin{aligned} 4s \ge &-2m \log \mu_k - \log V_k(R/\mu_k) + \log V_k(R) \\ &- \log V_k(\sqrt{n}) - \log V_n(5\sqrt{n}). \end{aligned}$$

Using the approximation [12],

$$V_d(r) = \frac{\pi^{d/2} r^d}{\Gamma(d/2+1)} \approx \frac{1}{\sqrt{\pi d}} \Big(\frac{2e\pi}{d}\Big)^{d/2} r^d,$$

and the fact that $\log V_d(rs) = \log V_d(r) + d \log s$, we find that

$$4s \ge -2m \log \mu_k + k \log \mu_k - \log V_k(\sqrt{n}) - \log V_n(5\sqrt{n}).$$

The last two terms add up to $O(n)$; therefore,

$$s \ge \frac{1}{8}(k - 2m) \log \lambda_k - O(n),$$

which establishes the spectral lemma. □

3. Range searching over a group. Let P be the point set consisting of the vertices of a $(\sqrt{n}-1) \times (\sqrt{n}-1)$ square grid. Each point x_i of P is weighted by some real number, which by abuse of notation we also call x_i. Our goal is to exhibit n halfplanes $h_1, \ldots, h_n$, and prove that computing the sum of the weights within each h_k, i.e., $\sum_{x_i \in h_k} x_i$, requires $\Omega(n \log n)$ time.

The model of computation is a straight-line program: each step performs a group operation of the form $z \leftarrow \alpha_1 x + \alpha_2 y$, where x and y are input weights or previously computed variables and $\alpha_i \in \{-1, 0, 1\}$. As a bonus, we also allow the use of close to $n/4$ instructions of the form $z \leftarrow f(x, y)$, where f is an arbitrary real function. The only requirement is that the same program should work for any assignment of real weights to the points. The analogy with the previous section is obvious. Let A be the $n \times n$ matrix whose kth row is the characteristic vector of $P \cap h_k$. By the spectral lemma, the lower bound of Theorem 1.1 follows directly from this lemma.

LEMMA 3.1. *There is a choice of n halfplanes, for which the matrix A of the corresponding set system is such that the kth largest eigenvalue of $A^\top A$ is $n^{\Omega(1)}$, for some $k \ge n/2 - \varepsilon n$, for any fixed $\varepsilon > 0$.*

Proof. We use a nonconstructive configuration of halfplanes. Scale down the square grid so that it fits within $[1/\sqrt{n}, 1 - 1/\sqrt{n}\,]^2$. Let ω be the motion-invariant measure for lines: we normalize ω to provide a probability measure for the lines crossing $[0,1]^2$. Given a halfplane h^+ bounded below by a nonvertical line h, consider the discrepancy function $f(h) \stackrel{\text{def}}{=} \sum_{x_i \in h^+} x_i$. A beautiful result of Alexander [1] (see [8, 10] for a simpler proof and various extensions) says that if $x_1 + \cdots + x_n = 0$, then[3]

$$\int f^2(h)\, d\omega(h) \gg \frac{1}{\sqrt{n}} \|x\|_2^2.$$

[3]We use the notation $\ll$ and $\gg$ to denote inequality up to a constant factor.

Subdivide the space of lines crossing $[0,1]^2$ into $N + O(n^2)$ regions within which the form $f(h)$ remains invariant. By choosing N large enough, say, $N = 2^n$, we can also ensure that the ω-area of N of these regions is exactly the same; call it σ, which is about $1/N$. The other $O(n^2)$ regions may have smaller areas. (Consider the arrangement in dual space to obtain this result.) Thus, the difference between integrating f^2 over the whole probability space and over the equal-area regions only is at most $O(n^2/N)\sup f^2$. Because $|f|$ cannot exceed

$$|x_1| + \cdots + |x_n| \le \sqrt{n}\ \|x\|_2,$$

the error is bounded by $O(n^3\|x\|_2^2/N)$. This provides us with a good discrete approximation of the L^2-norm of f. Indeed, let B be the $N \times n$ matrix whose rows are indexed by the N equal-area regions $\hat{\sigma}$ and are the characteristic vectors of the set of x_i's appearing in (the unique form) $f(h)$, for $h \in \hat{\sigma}$. We have

$$\Big|\ \|Bx\|_2^2 - \frac{1}{\sigma}\int f^2(h)\,d\omega(h)\ \Big| = O(n^3)\frac{\|x\|_2^2}{N\sigma}.$$

But $\sigma = 1/N \pm O(n^2/N^2)$, so

$$\Big|\ \|Bx\|_2^2 - N\int f^2(h)\,d\omega(h)\ \Big| = O(n^3\|x\|_2^2).$$

LEMMA 3.2.

$$\det B^\top B = \Omega\Big(N\Big/\sqrt{n}\Big)^{n-1}.$$

Proof. Let $\mu_1 \ge \cdots \ge \mu_n \ge 0$ be the eigenvalues of $B^\top B$ and let $\{v_i\}$ be an orthonormal eigenbasis, where v_i is associated with μ_i. We express $x = (\xi_1, \ldots, \xi_n)$ in the basis $\{v_i\}$. The solution space of the system of equations, $x_1 + \cdots + x_n = 0$ and $\xi_j = 0$ $(j < n-1)$, is of dimension at least 1. Since it lives in the (ξ_{n-1}, ξ_n) plane, it intersects the cylinder $\xi_{n-1}^2 + \xi_n^2 = 1$. For any point x of the intersection,

$$\|Bx\|_2^2 = \sum_{i=1}^{n}\mu_i\xi_i^2 = \mu_{n-1}\xi_{n-1}^2 + \mu_n\xi_n^2 \le \mu_{n-1}.$$

This implies that for this unit vector x,

$$\mu_{n-1} \ge N\int f^2(h)\,d\omega(h) - O(n^3\|x\|_2^2) \gg \frac{N}{\sqrt{n}} - O(n^3),$$

and hence,

$$\mu_{n-1} \gg \frac{N}{\sqrt{n}}. \tag{3}$$

We need a lower bound on the smallest eigenvalue, but almost any one will do. With N being large enough, we can always assume that for each point x_i there exist two lines, each represented by a distinct row of B, that pass right above and below x_i. The contribution of these two rows to $\|Bx\|_2^2$ is of the form $\Phi^2 + (\Phi + x_i)^2$, which is always at least $x_i^2/2$. It follows that $\|Bx\|_2^2 \ge \frac{1}{2}\|x\|_2^2$, and hence, $\mu_n \ge 1/2$. The lemma follows from (3) and the fact that $\det B^\top B$ is the product of the eigenvalues. □

Of course, the set system B is much too big. Indeed, the map $x \mapsto Bx$ is actually trivial to compute. We use a nonconstructive argument to prove the existence of a hard $n \times n$ set system A. By the Binet-Cauchy formula,[4]

$$\det B^\top B = \sum_{1 \le j_1 < \cdots < j_n \le N} \left| \det B \begin{pmatrix} j_1 & j_2 & \cdots & j_n \\ 1 & 2 & \ldots & n \end{pmatrix} \right|^2.$$

Therefore, there exists an $n \times n$ submatrix A of B such that

$$\begin{aligned} \det A^\top A &= \left| \det B \begin{pmatrix} j_1 & j_2 & \cdots & j_n \\ 1 & 2 & \ldots & n \end{pmatrix} \right|^2 \\ &\ge \binom{N}{n}^{-1} \det B^\top B = \Omega(1)^n \Big(\frac{n}{eN}\Big)^n \Big(\frac{N}{\sqrt{n}}\Big)^{n-1} \\ &\ge n^{n/2 - o(n)}, \end{aligned}$$

from which we find that

$$\log \det A^\top A \ge \Big(\frac{n}{2} - o(n)\Big) \log n. \tag{4}$$

By Morgenstern's result [18], it follows easily that in the absence of any help gates, halfplane range searching requires $\Omega(n \log n)$ operations. To be able to deal with help gates we must collect more information about the spectrum of $A^\top A$. Let $\lambda_1 \ge \cdots \ge \lambda_n \ge 0$ be the eigenvalues of $A^\top A$.

LEMMA 3.3. *For some constant $c > 0$,*

$$\lambda_k \le \frac{cn^2 \log(k+1)}{k}.$$

Proof. Let L be a set of representative lines whose corresponding upper halfplanes define the sets encoded in the rows of A. Subdivide the unit square into a regular $r \times r$ grid of lines (r to be chosen later), and throw in a random sample of r lines chosen among L. Form the arrangement of these $3r$ lines and triangulate it. With high probability, no triangle is cut by more than $cn(\log r)/r$ lines of L, and none contains more than cn/r^2 points, for some constant c (assumed large enough for future purposes). For each triangle write the linear constraint expressing that the sum of the x_i's within the triangle is null. This gives us a set of $k_0 \le cr^2$ linear constraints, called *canonical*. Assume that all are satisfied. Then, A can be rewritten in simpler form by means of a sparse matrix C. Specifically, by the zone theorem for line arrangements, we know that no line can cut more than cr triangles. Therefore, within the restriction to the constraint space, each row of A corresponds to a linear form with at most $c^2 n/r$ nonzero coefficients. Let C be the new matrix formed by those relevant entries. Note that no column (resp., row) of C contains more than $cn(\log r)/r$ (resp., $c^2 n/r$) ones.

It is a standard result [14] that the spectral norm of a matrix Q satisfies

$$\|Q\|_s^2 \le \Big(\max_i \sum_j |q_{ij}|\Big)\Big(\max_j \sum_i |q_{ij}|\Big),$$

[4]The notation refers to the matrix obtained by picking the rows indexed $j_1, \ldots, j_n$ in B.

and therefore the Rayleigh quotient $x^\top C^\top Cx/x^\top x$ ($x \neq 0$) is at most $c^3n^2(\log r)/r^2$. As a result, for any x satisfying the canonical constraints and $\|x\|_2 = 1$, we have

$$\|Ax\|_2^2 \leq \frac{c^3n^2 \log r}{r^2}. \tag{5}$$

Let $\{u_i\}$ be an orthonormal eigenbasis for $A^\top A$, where u_i is associated with λ_i. We express $x = (\xi_1, \ldots, \xi_n)$ in the basis $\{u_i\}$. The solution space of the system of equations consisting of $\xi_j = 0$ ($j \geq k_0 + 2$) and the canonical constraints is of dimension at least 1. It is embedded in the (k_0+1)-flat spanned by $(\xi_1, \ldots, \xi_{k_0+1})$, so it intersects the cylinder $\xi_1^2 + \cdots + \xi_{k_0+1}^2 = 1$. For any point x in the intersection, $\|x\|_2 = 1$ and

$$\|Ax\|_2^2 = \sum_{i=1}^{n} \lambda_i \xi_i^2 = \sum_{i=1}^{k_0+1} \lambda_i \xi_i^2 \geq \lambda_{k_0+1},$$

so by (5), $\lambda_{k_0+1} \leq c^3n^2(\log r)/r^2$. In the worst case, k_0 is proportional to r^2, so the lemma is true for any k large enough. For small k, we can use the straightforward bound $\lambda_k \leq n^2$. □

From the lemma we find that

$$\begin{aligned}
\log \det A^\top A &= \sum_{i=1}^{n} \log \lambda_i \\
&\leq (n-k) \log \lambda_k + \sum_{j=1}^{k} \log \frac{cn^2 \log(j+1)}{j} \\
&\leq (n-k) \log \lambda_k + k(2 \log n \\
&\qquad - \log k + \log \log k + c').
\end{aligned}$$

In view of (4) we find that we can set $k = n/2 - \varepsilon n$, for any fixed $\varepsilon > 0$, and still derive the lower bound $\log \lambda_k = \Omega(\log n)$, which proves Lemma 3.1. □

We conclude that halfplane range searching requires $\Omega(n \log n)$ time, even in the presence of close to $n/4$ free computations. Notice that over the reals the problem can be solved in linear time with only $2n-1$ free computations: the circuit is a tree of help gates whose leaves are the x_i's and whose root "collects" the vector $(x_1, \ldots, x_n)$ and encodes it as a real. Then, with another n help gates, we can distribute the correct n outputs: the total number of help gates is $2n-1$. This completes the proof of Theorem 1.1. □

Appendix. We prove (2): $M_r^+(n,m) \leq 3^n V_m(r)$, for $n > m$. Let $\mathcal{C}$ be the set of cubes counted by $M_r^+(n,m)$. Any cube $c \in \mathcal{C}$ has at least one $(n-m)$-face intersecting $F \cap B_n(O,r)$ in exactly one point (call it q_c). This face is supported by an $(n-m)$-flat which is specified by fixing exactly m integer coordinates. In other words, it is specified by an integral point p_c in the m-flat spanned by a set of m axes. Since by convexity such a point p_c corresponds to at most $2^{n-m}(n-m)$-faces and at most 2^m cubes can share the same $(n-m)$-face, counting the number of points p_c gives an upper bound on $M_r^+(n,m)$, up to a factor of 2^n. Of course, we can restrict the counting to the number of integral points that lie within any of the projections of $F \cap B_n(O,r)$ onto m-flats spanned by $x_{i_1}, \ldots, x_{i_m}$. Furthermore, we can discount projections that map F to a flat of dimension less than m (because q_c is uniquely defined). Let $E_{i_1,\ldots,i_m}$ be the ellipsoid obtained by projecting $F \cap B_n(O,r)$ onto

the flat $(x_{i_1}, \ldots, x_{i_m})$. We say that a point p_c is *peripheral* if it is the upper corner (upper with regard to all m dimensions) of a cube not fully contained in the projected ellipsoid, and we let N denote the total number of peripheral points. We have

$$M_r^+(n,m) \le 2^n N + 2^n \sum_{1 \le i_1 < \cdots < i_m \le n} \operatorname{vol} E_{i_1,\ldots,i_m}. \tag{6}$$

Let $v_1, \ldots, v_m$ be an orthonormal basis for F and let U be the $n \times m$ matrix whose columns are the v_i's. (Note that $U^\top U$ is the identity matrix.) The determinant of the $m \times m$ submatrix of U specified by the rows $(i_1, \ldots, i_m)$ is equal, in absolute value, to the volume of $E_{i_1,\ldots,i_m}$ divided by $V_m(r)$. (We need only sketch the proof of this simple fact: lift to F the principal vectors of the ellipsoid and scale them to unit length; this provides an orthonormal basis for F that satisfies the claim. Because the basis $\{v_i\}$ can be derived from it by a unitary transformation within F, the claim follows.) By the Binet-Cauchy formula, the determinant of $U^\top U$ can be expressed as

$$\sum_{1 \le i_1 < \cdots < i_m \le n} \left| \det U \begin{pmatrix} i_1 & i_2 & \ldots & i_m \\ 1 & 2 & \ldots & m \end{pmatrix} \right|^2,$$

and by Cauchy-Schwarz,

$$\sum_{i_1,\ldots,i_m} \operatorname{vol} E_{i_1,\ldots,i_m} \le V_m(r) \binom{n}{m}^{1/2} \sqrt{\det U^\top U} = V_m(r) \binom{n}{m}^{1/2}.$$

On the other hand, in the flat $(x_{i_1}, \ldots, x_{i_m})$, every cube of the unit lattice that has a vertex in $E_{i_1,\ldots,i_m}$ and that intersects $\partial E_{i_1,\ldots,i_m}$ does so along at least one edge: at most four such edges can be collinear, and no more than 2^{m-1} cubes can charge the same edge. Projecting $E_{i_1,\ldots,i_m}$ by dropping one coordinate gives an $(m-1)$-ellipsoid whose integral points are contained in an $(m-1)$-ball of radius r. Within a factor of 2^{m+1}, the total number of such points is an upper bound on N. Each such point is the upper corner of a distinct unit cube in a ball of radius $r + \sqrt{m-1}$; therefore, $N \le m 2^{m+1} \binom{n}{m} V_{m-1}(r + \sqrt{m-1})$, and by (6),

$$M_r^+(n,m) \le m\, 4^n \binom{n}{m} V_{m-1}(r + \sqrt{m-1}) + 2^n \binom{n}{m}^{1/2} V_m(r).$$

As r goes to infinity the second term becomes dominant, and for n large enough, (2) follows.

Acknowledgments. I wish to thank Dick Lipton, Ran Raz, Avi Wigderson, and Andy Yao for helpful discussions. I also thank the referees for many useful comments and suggestions.

REFERENCES

[1] R. Alexander, *Geometric methods in the study of irregularities of distribution*, Combinatorica, 10 (1990), pp. 115–136.

[2] N. Alon, M. Karchmer, and A. Wigderson, *Linear circuits over GF(2)*, SIAM J. Comput., 19 (1990), pp. 1064–1067.

[3] A. Amir, R. Beigel, and W. Gasarch, *Some connections between bounded query classes and nonuniform complexity*, 5th Annual IEEE Structure in Complexity Theory Conference, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 232–243.

[4] J. Beck and W. W. L. Chen, *Irregularities of Distribution*, Cambridge Tracts in Mathematics 89, Cambridge University Press, Cambridge, 1987.
[5] R. B. Boppana and M. Sipser, *The complexity of finite functions*, in Handbook of Theoretical Computer Science, Vol. A, Algorithms and Complexity, MIT Press/Elsevier, Cambridge/New York, 1990, pp. 757–804.
[6] J. Y. Cai, *Lower bounds for constant depth circuits in the presence of help bits*, in Proc. 30th Annual Symp. Foundation Comput. Sci., (FOCS), IEEE Computer Society Press, Los Alamitos, CA, 1989, pp. 532–537.
[7] B. Chazelle, *Lower bounds on the complexity of polytope range searching*, J. Amer. Math. Soc., 2 (1989), pp. 637–666.
[8] B. Chazelle, *Geometric discrepancy revisited*, in Proc. 34th Annual IEEE Symp. Foundation Comput. Sci., (FOCS), IEEE Computer Society Press, Los Alamitos, CA, 1993, pp. 392–399.
[9] B. Chazelle, *Computational geometry: A retrospective*, in Computing in Euclidean Geometry, 2nd ed., D.-Z. Du and F. Hwang, eds., World Scientific Press, River Edge, NJ, 1995, pp. 22–46.
[10] B. Chazelle, J. Matoušek, and M. Sharir, *An elementary approach to lower bounds in geometric discrepancy*, Discrete Comput. Geom., 13 (1995), pp. 363–381.
[11] M. L. Fredman, *Lower bounds on the complexity of some optimal data structures*, SIAM J. Comput., 10 (1981), pp. 1–10.
[12] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, New York, 1988.
[13] J. von zur Gathen, *Algebraic complexity theory*, Ann. Rev. Comput. Sci., 1988, pp. 317–347.
[14] P. Lancaster and M. Tismenetsky, *The Theory of Matrices*, 2nd ed., Academic Press, New York, 1985.
[15] J. Matoušek, *Range searching with efficient hierarchical cuttings*, Discrete Comput. Geom., 10 (1993), pp. 157–182.
[16] J. Matoušek, *Geometric range searching*, Tech. Report B-93-09, Free Univ. Berlin, 1993.
[17] K. Mehlhorn, *Data Structures and Algorithms 3: Multidimensional Searching and Computational Geometry*, Springer-Verlag, Heidelberg, 1984.
[18] J. Morgenstern, *Note on a lower bound of the linear complexity of the fast Fourier transform*, J. ACM, 20 (1973), pp. 305–306.
[19] K. Mulmuley, *Computational Geometry: An Introduction Through Randomized Algorithms*, Prentice–Hall, Englewood Cliffs, NJ, 1994.
[20] V. Strassen, *Algebraic complexity theory*, in Handbook of Theoretical Computer Science, Vol. A, Algorithms and Complexity, MIT Press/Elsevier, Cambridge/New York, 1990, pp. 633–672.

SIAM J. COMPUT.
Vol. 27, No. 2, pp. 557–582, April 1998
© 1998 Society for Industrial and Applied Mathematics
012

INCREMENTAL STRING COMPARISON*

GAD M. LANDAU†, EUGENE W. MYERS‡, AND JEANETTE P. SCHMIDT§

Abstract. The problem of comparing two sequences A and B to determine their longest common subsequence (LCS) or the edit distance between them has been much studied. In this paper we consider the following incremental version of these problems: given an appropriate encoding of a comparison between A and B, can one incrementally compute the answer for A and bB, and the answer for A and Bb with equal efficiency, where b is an additional symbol? Our main result is a theorem exposing a surprising relationship between the dynamic programming solutions for two such "adjacent" problems. Given a threshold k on the number of differences to be permitted in an alignment, the theorem leads directly to an $O(k)$ algorithm for incrementally computing a new solution from an old one, as contrasts the $O(k^2)$ time required to compute a solution from scratch. We further show, with a series of applications, that this algorithm is indeed more powerful than its nonincremental counterpart. We show this by solving the applications with greater asymptotic efficiency than heretofore possible. For example, we obtain $O(nk)$ algorithms for the longest prefix approximate match problem, the approximate overlap problem, and cyclic string comparison.

Key words. string matching, edit-distance, dynamic programming

AMS subject classification. 68P99

PII. S0097539794264810

1. Introduction.

Sequence comparison is an extensively studied topic. Applications are numerous and include file comparison [HS-77], spelling correction [HD-80], information retrieval [WM-92], and searching for similarities among biosequences [NW-70, Se-80, SW-81]. Given string $A = a_1a_2a_3 \ldots a_m$ and $B = b_1b_2b_3 \ldots b_n$, one seeks an *alignment* between the two strings that exposes their similarity. An alignment is any pairing of symbols subject to the restriction that if lines were drawn between paired symbols as in Figure 1 below, the lines would not cross. Scores are assigned to alignments according to the concept of similarity or difference required by the context of the application, and one seeks alignments of optimal score [WF-74].

While for applications such as comparing protein sequences the methods of scoring can involve arbitrary scores for symbol pairs and for gaps of unaligned symbols, in many other contexts simple unit cost schemes suffice. Two of these, the *longest common subsequence (LCS)* and the *edit-distance* measures, have been studied extensively within computer science, and the unit cost nature of the scoring provides combinatorial leverage not found in the more general framework [Hi-77, HS-77, NKY-82, Uk-85a, Uk-85b, My-86a, LV-89, GP-90]. In the edit distance problem, each mismatched aligned pair and unaligned symbol is called a *difference* and scores 1. All

*Received by the editors March 21, 1994; accepted for publication (in revised form) March 27, 1996.

http://www.siam.org/journals/sicomp/27-2/26481.html

†Department of Computer Science, Polytechnic University, 6 MetroTech, Brooklyn, NY 11201 (landau@pucs2.poly.edu). The research of this author was partially supported by NSF grant CCR-9305873 and the New York State Science and Technology Foundation Center for Advanced Technology.

‡Department of Computer Science, University of Arizona, Tucson, AZ 85721 (gene@cs.arizona.edu). The research of this author was partially supported by NLM grant LM-04960, NSF grant CCR-9002351, and DOE grant DE-FG05-91ER61132.

§Department of Computer Science, Polytechnic University, 6 MetroTech, Brooklyn, NY 11201 (jps@pucs4.poly.edu). The research of this author was partially supported by NSF grants CCR-9305873 and HRD-9627109 and by the New York State Science and Technology Foundation Center for Advanced Technology.

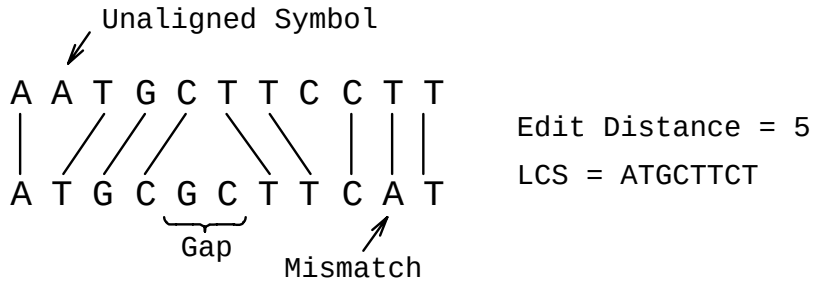


FIG. 1. *An alignment between two strings.*

pairs of equal aligned characters score 0. One seeks an alignment that minimizes the score or number of differences and this minimal score, $ED(A, B)$, is called the *edit distance* between A and B. Conversely, in the LCS problem matched pairs score 1, mismatches and unaligned symbols score 0, and the goal is to find an alignment of maximum score $LCS(A, B)$. In this case, the sequence of matched characters is a subsequence common to both sequences and is of maximum length, hence the name longest common subsequence. Figure 1 illustrates these measures.

Although certainly complementary, the LCS and edit distance problems are not formal duals. The LCS problem is equivalent to finding the minimum edit distance, *where mismatches are not allowed*, or equivalently, where a mismatch scores 2, so that it is no better than leaving the two characters unaligned. The reader may wish to also verify that the edit distance problem is equivalent to the following "LCS-like" problem: find an alignment of maximum score where matches score 1, unaligned characters score 0, and *mismatches score* $\frac{1}{2}$.

In this paper we consider the following incremental version of the sequence comparison problem: given a solution for the comparison of A and B, can one incrementally compute a solution for A versus bB, where b is an additional symbol prepended to B? By *solution* we mean some encoding of a relevant portion of the traditional dynamic programming matrix D computed in comparing A and B. D is an $(m + 1) \times (n + 1)$ matrix, where entry $D[i, j]$ is the best score for the problem of comparing A^i with B^j, and A^i is the prefix, $a_1 a_2 \ldots a_i$, of A's first i symbols. As will be seen in detail later, the data-dependencies of the fundamental recurrence, used to compute an entry $D[i, j]$, is such that it is easy to extend D to a matrix D' for A versus Bb by computing an additional column. However, efficiently computing a solution for A versus bB given D is much more difficult, in essence requiring one to work against the "grain" of these data-dependencies. The further observation that the matrix for A versus B, and that the matrix for A versus bB can differ in $O(mn)$ entries, suggests that the relationship between such adjacent problems is nontrivial.

One might immediately suggest that by comparing the reverse of A and B, prepending symbols becomes equivalent to appending symbols, and so the problem, as stated, is trivial. But in this case, we would ask for the delivery of a solution for A versus Bb. The point is that our method allows one to append *and* prepend symbols to A and/or B in any order, efficiently solving one problem from the previous one. More formally, given an initial solution $S^{(0)}$ for $A^{(0)} = A$ versus $B^{(0)} = B$, and a sequence of operations $op^{(t)}$ that either prepend or append a single symbol to $A^{(t-1)}$ or $B^{(t-1)}$ to produce $A^{(t)}$ and $B^{(t)}$, we can deliver the corresponding sequence of solutions $S^{(t)}$ with linear efficiency. Moreover, we can do so online, i.e., the sequence of

operations need not be known in advance. To keep matters simple, however, we will focus on the core problem of computing a solution for A versus bB, given a "forward" solution for A versus B. A "forward" solution of the problem contains an encoding of a comparison of all (relevant) prefixes of A with all (relevant) prefixes of B. It turns out that the ability to efficiently prepend a symbol to B when given all the information contained in a "forward" solution allows one to solve the applications given in section 4 with greater asymptotic efficiency than heretofore possible.

As an example of such application, consider the problem of approximate string matching within k differences: find *all* substrings of a text B of length n whose edit distance from a query A is not greater than threshold k. With our $O(k)$ incremental version of the well-known $O(n + k^2)$ greedy algorithm for edit distance, the problem can be solved in $O(nk)$ time by starting with an empty text and building it up from the right, one character at a time. Proceeding formally, let B_l denote the *suffix* of B starting at its $l + 1$st symbol, $b_{l+1}b_{l+2} \ldots b_n$. Begin the search by building the trivial k-thresholded solution for A versus B_n $(= \varepsilon)$. Then incrementally compute the solutions to A versus B_l for $l = n - 1, n - 2, n - 3 \ldots 0$ in $O(k)$ time per step. While the overall time, $O(nk)$, is no better than previous results, [LV-89, GP-90], the algorithm is superior in that for each left index l it reports *all* right indexes r, (and for each right index r it hence reports *all left* indexes l), delimiting a substring $B_l^r = b_{l+1}b_{l+2} \ldots b_r$ that matches A within k differences. For each such match-pair (l, r), the algorithm delivers the number of differences, $ED(A, B_l^r)$, in the match. Previous algorithms either report for each right index r, the matches with the smallest number of differences ending at r (the "forward" solution), or report the matches with the smallest number of differences starting at l (the "backward" solution). In our solution at each potential left-index l, the entire k-thresholded solution for A versus B_l is available and can be examined in $O(k)$ time to find any corresponding right-indices and their match score. In addition, if A does not match any prefix of B_l with at most k differences, we can report the longest prefix of A that matches a prefix of B_l with k differences. If desired, one can also build an $O(nk)$ table $T[0 \ldots n][-k \ldots k]$ during the search where $T[l][r - (l + m)]$ equals $ED(A, B_l^r)$, if (l, r) delimits a k-match (i.e., a match with at most k differences) and $k + 1$ otherwise. The table T is a record of all k-matches found, as r must be in the interval $[l + m - k, l + m + k]$ if (l, r) delimits a k-match. It is impossible for previous string-matching algorithms to be augmented (1) to report, at each position of B, the longest prefix of A matching with k differences, (2) to build T, or, equivalently, (3) to report *all* k-matching substrings and their match distances in $O(nk)$ time. Such a capability is essential, for example, if one is searching for the best match under a scoring criterion that is a complex function of the length and number of differences in the match.

The algorithmic results of this paper hinge on what we find to be the rather surprising fact that there are exploitable relationships between the dynamic programming solutions of adjacent problems computed by several well-known comparison algorithms for LCS and edit distance. Throughout the paper we will focus on formulating incremental versions of the well-known $O(n + k^2)$ greedy algorithms for finding the edit distance and the LCS between two sequences [My-86a, LV-88]. A similar and somewhat simpler incremental version [My-86b] also holds for the $O(r \log n)$ Hunt–Szymanski algorithm [HS-77]. After a presentation of preliminary concepts and a review of the $O(n + k^2)$ greedy algorithm in section 2, we present the main theorem exposing the relationship between adjacent solutions and sketch an incremental algorithm based on it in section 3. Section 4 then presents four applications of incremental algorithms in order to demonstrate the power of such algorithms.

		A	G	G	A	T	A	T	T	A	
	0	1	2	3	4	5	6	7	8	9	0
A	1	0	1	2	3	4	5	6	7	8	1
T	2	1	1	2	3	3	4	5	6	7	2
G	3	2	1	1	2	3	4	5	6	7	3
G	4	3	2	1	2	3	4	5	6	7	4
T	5	4	3	2	2	2	3	4	5	6	5
A	6	5	4	3	2	3	2	3	4	5	6
T	7	6	5	4	3	2	3	2	3	4	7
A	8	7	6	5	4	3	2	3	3	3	8
	0	1	2	3	4	5	6	7	8	9	

FIG. 2. *A sample dynamic programming matrix.*

2. Preliminaries.

2.1. The dynamic programming algorithm. Consider the problem of computing the edit distance between strings $A = a_1a_2 \ldots a_m$ and $B = b_1b_2 \ldots b_n$ where without loss of generality we assume that $m \leq n$ hereafter. The well-known dynamic programming algorithm [NW-70, WF-74] computes an $(m+1) \times (n+1)$ *edit-distance matrix* $D[0 \ldots m][0 \ldots n]$, where entry $D[i,j]$ is the edit distance $ED(A^i, B^j)$ between the prefixes A^i and B^j of A and B, and where $A^i = A^i_0 = a_1 \ldots a_i$ and $B^j = B^j_0 = b_1 \ldots b_j$ as defined in the introduction. Figure 2 gives an example of the matrix $D[0 \ldots 8, 0 \ldots 9]$ for $A = ATGGTATA$ versus $B = AGGATATTA$. The edit distance between A and B is given in entry $D[8,9]$ which is 3. Conceptually we think of D as an $(m+1) \times (n+1)$ grid of *points* (i,j) to which we assign value $D[i,j]$.

A best alignment between $A^i = a_1 \ldots a_i$ and $B^j = b_1 \ldots b_j$ must either (1) leave a_i unaligned and optimally align A^{i-1} and B^j, (2) leave b_j unaligned and optimally align A^i and B^{j-1}, or (3) align a_i and b_j and optimally align A^{i-1} and B^{j-1}. This observation leads immediately to the following fundamental dynamic programming recurrence.

LEMMA 2.1. *For all* $i + j > 0$,

$$D[i,j] = \min \left\{ \begin{array}{ll} D[i-1,j]+1 & \text{if } i > 0 \\ D[i,j-1]+1 & \text{if } j > 0 \\ D[i-1,j-1]+\delta_{a_i,b_j} & \text{if } i,j > 0 \end{array} \right\}$$

where $\delta_{a,b}$ *is* 1 *or* 0 *depending on whether or not* $a = b$*, respectively.*

Coupled with the obvious boundary condition that $D[0,0] = 0$, this recurrence can be used to efficiently compute the $O(mn)$ entries of the matrix D in an order of i and j that observes the data-dependencies of the recurrence, i.e., an order that computes $D[i-1,j]$, $D[i,j-1]$, and $D[i-1,j-1]$ before $D[i,j]$. Traditionally, the lexicographical order of (i,j) (which clearly observes data-dependencies) is used to determine the value of the equation in Lemma 2.1 in $O(1)$ time. An algorithm based on this recurrence thus takes $O(mn)$ time to compute the value of every point of D. The edit distance $ED(A,B)$ is delivered in $D[m,n]$.

The matrix D has a number of useful monotonicity properties with respect to diagonals that are essential to our result.

DEFINITION 2.2. *Diagonal* d *is the list of all points* (i,j) *for which* $j = i + d$.

Note that with this definition the lowest, leftmost diagonal is numbered $-m$ and the highest, rightmost diagonal is numbered n. The first essential property is that values are nondecreasing along diagonals and never increase by more than one.

LEMMA 2.3 (see [Uk-85a]). *For all points* (i, j)*:* $D[i, j] - D[i-1, j-1] \in \{0, 1\}$.

The second essential property is that adjacent values in adjacent diagonals never differ by more than one.

LEMMA 2.4 (see [Uk-85b]). *For all points* (i, j)*:* $D[i, j]-D[i-1, j], D[i, j]-D[i, j-1] \in \{-1, 0, 1\}$.

2.2. The greedy algorithm. In the mid-80s [NKY-82, Uk-85a] came upon the idea of computing the points of the matrix D in an order dictated by a greedy approach, (i.e., to compute the values in nondecreasing order), instead of the lexicographic order of (i, j). In this way entries whose values are 0 are computed first, then those whose values are 1, and then 2, and so on until either (i) some threshold k is reached, or (ii) the value $D[m, n]$ is determined.

By Lemma 2.1, $D[0, j] = j$ for $j = 0, \ldots, n$, and $D[i, 0] = i$ for $i = 0, \ldots, m$. By Lemma 2.3, the values along a diagonal are nondecreasing, so that $D[0, d]$ (resp., $D[d, 0]$) are the smallest values on diagonal d, (resp., $-d$). We conclude that all values not greater than k are on diagonals $-k, -k+1, \ldots, k$ in D, and there are therefore at most $(2k + 1)m$ entries with such values. This idea gives rise to an $O(km)$ algorithm to determine the k-thresholded edit distance of A and B.

The algorithm can be realized by a modified version of Dijkstra's shortest path algorithm on a directed, weighted *edit graph* $G = (V, E)$. V consists of the set of all points (i, j) of D. For each point (i, j) there is an edge into it from $(i-1, j)$ weighted 1, another from $(i, j-1)$ also weighted 1, and a third edge from $(i-1, j-1)$ weighted δ_{a_i, b_j}. For points along the left and upper boundaries of D, there is an edge from a predecessor of (i, j) only if the predecessor exists. Observe that the edge weights are chosen in exact correspondence with Lemma 2.1, and that $D[i, j]$ is the length of the shortest path from $(0, 0)$ to (i, j) in this edit graph. The algorithm stops when either (i) the distance to (m, n) is determined, or (ii) the first node with distance greater than k from $(0, 0)$ is reached. Since the lengths of all paths are integers in $[0 \ldots k]$, and the outdegree of each node in the graph is at most 3, a standard modification of Dijkstra's algorithm will run in $O(km)$ time and space.

Ukkonen [Uk-85a] noticed that it suffices to determine, for all h, the last h on each diagonal d of the matrix D, as this determines all other values in D by Lemma 2.3. More precisely, let $L^h(d)$ denote the largest row index of a point on diagonal d that has value h.

DEFINITION 2.5. $L^h(d) = \max\{\, i \,:\, D[i, i+d] = h \,\}$.

Figure 3 illustrates this definition by labeling just these *furthest h-points* with their values. Note that the row number, $i = L^h(d)$, of the furthest h-point on diagonal d identifies the point itself as $(i, i + d)$. So henceforward, we will liberally use $L^h(d)$ to denote either the point $(i, i + d)$ or the row i. The interpretation of $L^h(d)$ will be obvious from context.

As observed earlier, there are no h-points outside diagonals $[-h \ldots h]$. Thus it suffices to compute for each value h, the *h-wave*.

DEFINITION 2.6. $L^h = \langle L^h(-h), L^h(-h+1), \ldots, L^h(0), \ldots, L^h(h-1), L^h(h)\rangle$.

That is, L^h is the orderered list consisting of the $2h + 1$ furthest h-points in diagonals $-h$ up to h.

The algorithm then computes wave L^h, for $h = 0, 1, 2, \ldots$, until either (i) a wave e is computed for which $L^e(n-m) = m$, or (ii) wave L^k is computed in the case that the algorithm is thresholded by k. In the event that termination is by condition (i) it follows that $ED(A, B) = D[m, n] = e$.

Note that the highest values on some of the diagonals may be less than h. In Figure 3 for example, the highest value in diagonal -2 is 2 (in $D[8, 6]$), and $L^3(-2)$

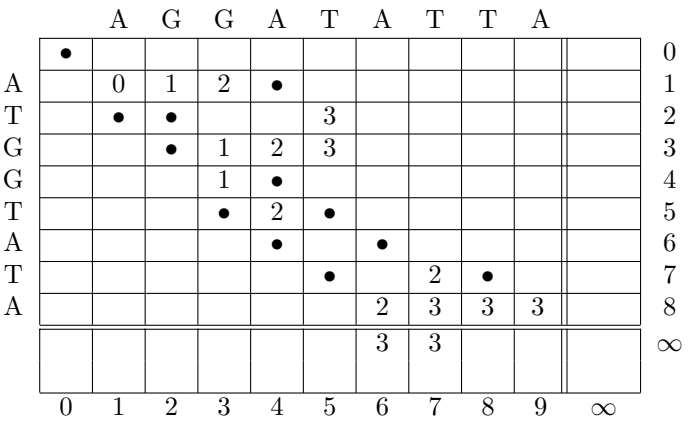


FIG. 3. *The h-waves of the matrix of Figure* 2.

therefore does not exist, as there is no 3-point in diagonal -2. We handle this overflow by adding a "dummy" point $L^h(d)$ with value ∞ to wave h, whenever no real point $L^h(d)$ exists, so that L^h always has $2h+1$ points. In addition, when the extreme point on diagonal d has value h, it is also convenient to set $L^h(d)$ to ∞ whenever (1) this extreme point is $(m, m+d)$, $D[m, m+d] = h$, and $D[m, m+d+1] = h-1$, or when (2) this extreme point is $(n-d, n)$, $D[n-d, n] = h$, and $D[n-d+1, n] = h-1$. Intuitively, this is justified because if we were to extend the matrix D with one more row and column, then $D[m+1, m+d+1]$ (resp., $D[n-d+1, n+1]$) would still always have value h, regardless of how we interpret the "dummy" character associated with that row (resp., column). This is shown in Figure 3, where $L^3(-3)$ is shown outside the boundary of the matrix and hence assigned ∞, although the preceding point on that diagonal, $D[8, 5]$, also has value 3.

Given wave L^{h-1}, wave L^h is computed from it by induction. Consider $i = L^{h-1}(d)$, the furthest $(h-1)$-point in diagonal d, and a corresponding optimal alignment between A^i and B^j, where $j = i+d$. The alignment involves $h-1$ differences and $a_{i+1} \neq b_{j+1}$, as otherwise $(i+1, j+1)$ would also be an $(h-1)$-point (a contradiction). Observe that in this case this alignment can be maximally extended, with one additional difference, in the following three ways: (1) leave a_{i+1} unaligned and then align a_{i+1+q} with b_{j+q} for $q > 0$ as long as the symbols are equal, (2) leave b_{j+1} unaligned and then align a_{i+q} with b_{j+1+q} for $q > 0$ as long as the symbols are equal, and (3) mismatch a_{i+1} with b_{j+1} and then align a_{i+1+q} with b_{j+1+q} for $q > 0$ as long as the symbols are equal. In each case, we visualize the alignment of equal symbols as "sliding down" the relevant diagonal: $d-1$ in case (1), $d+1$ in case (2), and d in case (3).

We capture such a substring of equal characters, resulting in a slide down a diagonal, with the following definition.

DEFINITION 2.7. $Slide_d(i) = \max\{\ q\ :\ A_i^q = B_{i+d}^{q+d}\ \}$.

That is, $Slide_d(i)$ corresponds to a slide in diagonal d starting on row i. In order to correctly handle the cases where $L^h(d)$ is or becomes ∞, we define $Slide_d(i) = \infty$, when $i > m$ or $i + d > n$. Note that by definition $Slide_d(i) = i$ when $i = m$ or $i + d = n$.

The gist of earlier papers was a proof that the furthest point reached in diagonal d over the relevant extensions from the points $L^{h-1}(d-1)$, $L^{h-1}(d)$, and

$L^{h-1}(d+1)$ on wave $h-1$ is the point $L^h(d)$.[1] Combining this with the identity, $\max\{Slide_d(a), Slide_d(b)\} = Slide_d(\max\{a,b\})$, leads to the following recurrence for a furthest h-point in terms of furthest $(h-1)$-points.

LEMMA 2.8.

$$\textit{For all } h > 0, L^h(d) = Slide_d\left(\max\left\{\begin{array}{ll} L^{h-1}(d+1)+1 & \textit{if } d < h-1 \\ L^{h-1}(d)+1 & \textit{if } -h < d < h \\ L^{h-1}(d-1) & \textit{if } d > -h+1 \end{array}\right\}\right).$$

Using Lemma 2.8 one can immediately compute the $2h+1$ points of wave L^h, given the $2h-1$ points of wave L^{h-1}. The induction of the algorithm is started by observing that wave L^0 is the single point $L^0(0) = Slide_0(0)$.

In Figure 3 the furthest points of waves 0 through 3 are indicated by the placement of their value at their location, and the solid circles annotate points that the function *Slide* extends through to reach a furthest point. Note that many of the points in the matrix D of Figure 2, that have value 3 or less, are not marked in Figure 3.

The time complexity of the greedy algorithm depends on the efficiency with which the function *Slide* is realized. When *Slide* is computed by a brute-force comparison of the relevant characters, computing $s = Slide_d(i)$ takes $O(s-i)$ time, resulting in $O(km)$ total time. However, Myers [My-86a] has shown that when one of the strings, say A, is a random string[2] then the algorithm takes $O(m+k^2)$ *expected time*. This result is true even if B is chosen so as to maximize the time spent by the *Slide* function.

2.3. An $O(n+k^2)$ algorithm. The worst-case time of the previous algorithm is improved to $O(n+k^2)$ by computing *Slide* in constant time [My-86a, LV-88].

In a preprocessing step one computes a suffix tree [Wn-73, Mc-76] of the string $AxBy = a_1a_2\ldots a_m x b_1 b_2 \ldots b_n y$ where $x \neq y$ are two symbols not in the alphabets of A and B. One further preprocesses this suffix tree using any of the algorithms [HT-84, SV-88, BV-93], to allow any LCA (least common ancestor) query over the tree to be answered in $O(1)$ time. This preprocessing takes $O(n)$ time.[3]

For given indices i, j the *Slide* function must return the largest q for which $a_i \ldots a_{i+q} = b_j \ldots b_{j+q}$. The key observation is that this q can be retrieved from the suffix tree described above with an LCA query in $O(1)$ time. Specifically, it has been shown that $Slide_d(i) = depth(LCA(leaf(i), leaf(m+1+i+d)))$ where $leaf(t)$ is the leaf in the suffix tree for suffix $(AxBy)_t$, $LCA(u,v)$ is the LCA of vertices u and v, and $depth(v)$ is the length of the string that labels the path from the root of the suffix tree to vertex v.

As noted earlier, wave L^h has $2h+1$ points, each of which can now be computed in $O(1)$ time. Thus it takes $O(k^2)$ worst-case time to compute the $(k+1)^2$ points in waves L^0 through L^k after the initial $O(n)$ preprocessing. This results in an $O(n+k^2)$ worst-case time algorithm.

3. The central theorem and basic algorithm. Given two strings $A = a_1a_2\ldots a_m$ and $B = b_1b_2\ldots b_n$, we now show how to compute the $k+1$ waves $L^0_{new}, \ldots, L^k_{new}$ of the edit-distance matrix $D_{new}[i,j]$ of A and bB, when given the $k+1$ waves $L^0_{old}, \ldots, L^k_{old}$ of the edit-distance matrix $D_{old}[i,j]$ of A versus B.

---

[1] This is not immediately obvious as indicated by the fact that this property is *not* true for more general, weighted-cost comparison models.

[2] Specifically, the result of Bernoulli trials over a finite distribution.

[3] $O(n \log \Sigma)$ time if the alphabet size, Σ, is not considered fixed.

D_{new}, when viewed as a leftward extension of D_{old}, is an $(m+1)\times(n+2)$ matrix $D_{new}[0\ldots m][-1\ldots n]$, with main diagonal labeled -1. Labeling the columns in D_{new} from -1 to n (as opposed to $0\ldots n+1$) has the advantage that entry $D_{new}[i,j]$ corresponds to the edit distance, $ED(A^i, bB^j)$ between the prefixes A^i and bB^j of A and bB, establishing the correspondence of the points $D_{new}[i,j]$ and $D_{old}[i,j]$. Furthermore, for any index pair (i,j) that is valid in both D_{old} and D_{new}, $Slide_{j-i}(i)$ is the same in both matrices.

Since the waves for D_{new} are computed after those for D_{old} have been computed, we shall refer to L^h_{old} and D_{old} as the old h-wave and old matrix and to L^h_{new} and D_{new} as the new h-wave and matrix. We show that L^h_{new} is composed of a concatenation of a prefix of L^{h+1}_{old}, a sublist of L^h_{old}, a suffix of L^{h-1}_{old}, and at most two points p_1 and p_2, separating the sublists of L_{old}, not included in any of the old waves L^{h-1}_{old}, L^h_{old}, L^{h+1}_{old}. Furthermore, the two points p_1 and p_2 can be computed in $O(1)$ time and the entire list L^h_{new} can be pasted together from the lists L^{h+1}_{old}, L^h_{old}, L^{h-1}_{old} in $O(1)$ time.

The following five observations define the concepts and terminology needed to formulate and prove our central theorem.

The first observation relates the values in the matrices D_{old} and D_{new}. We note that any alignment between A^i and B^j with k differences can easily be used to obtain an alignment between A^i and bB^j with at most $k+1$ differences, and any alignment between A^i and bB^j with k differences can easily be used to obtain an alignment between A^i and B^j with at most $k+1$ differences. This implies the following.

Observation 1.

$$\forall (i,j) \in [0\ldots m, 0\ldots n],\ \ D_{old}[i,j]-1 \le D_{new}[i,j] \le D_{old}[i,j]+1$$
$$\forall i \in [0\ldots m],\ \ D_{new}[i,-1] = i.$$

D_{old} is not defined on $(i,-1)$. The point following $(i,-1)$ on diagonal $-i-1$ is $(i+1,0)$ and $D_{old}[i+1,0] = i+1$.

The second observation is an immediate consequence of the recurrence for computing $L^h(d)$ from $L^{h-1}(d-1)$, $L^{h-1}(d)$, and $L^{h-1}(d+1)$ given in Lemma 2.8. Recall that we identify the points on a given diagonal by their row number. Hence, an inequality $p > q$ (even if p and q are not on the same diagonal) is always interpreted as "the row number of p is higher than the row number of q."

Observation 2. If the three points of wave L^{g-1}_{old} on diagonals $d-1$, d, and $d+1$ are all less than or equal to (resp., greater than or equal to) the three points on those diagonals for L^{h-1}_{new}, then $L^g_{old}(d) \le L^h_{new}(d)$ (resp., $L^g_{old}(d) \ge L^h_{new}(d)$). If the three points on the two waves are all equal, then clearly $L^g_{old}(d) = L^h_{new}(d)$. Furthermore, if $\max\{L^{g-1}_{old}(d+1)+1, L^{g-1}_{old}(d)+1, L^{g-1}_{old}(d-1)\}$ is less than or equal to $\max\{L^{h-1}_{new}(d+1)+1, L^{h-1}_{new}(d)+1, L^{h-1}_{new}(d-1)\}$, then $L^g_{old}(d) \le L^h_{new}(d)$.

In the following it will be important to distinguish how a given point $L^h(d)$ got its value in the equation of Lemma 2.8. We will say that

$L^h(d)$ was *obtained from above* iff $L^h(d) = Slide_d(L^{h-1}(d+1)+1)$.

$L^h(d)$ was *obtained diagonally* iff $L^h(d) = Slide_d(L^{h-1}(d)+1)$.

$L^h(d)$ was *obtained from the left* iff $L^h(d) = Slide_d(L^{h-1}(d-1))$.

Note that a point can be obtained in more than one way.

Our third observation compares the scope of the new h wave with the scope of the old $h-1$, h, and $h+1$ wave.

Observation 3. Wave $L^h_{new} = \langle L^h_{new}(-h-1), \ldots, L^h_{new}(h-1)\rangle$, and so it has a point on each of diagonals $-h-1$ through $h-1$ and only these diagonals. Similarly, observe that L^g_{old} has a point on each (and only) the diagonals $-g$ through g. Thus

the leftmost diagonal of L^h_{new}, $-h-1$, is also the leftmost diagonal of L^{h+1}_{old} and both L^h_{old} and L^{h-1}_{old} do not have a point on this diagonal or to the left of this diagonal. On the other hand, the rightmost diagonal of L^h_{new}, $h-1$ is also the rightmost diagonal of L^{h-1}_{old}, and both L^h_{old} and L^{h+1}_{old} do have a point on this diagonal as well as to the right of this diagonal.

Crucial to our central result is the idea of the *key value* of a point, $p = L^h_{new}(d)$, on a new wave, which describes p's position relative to points in the old waves. Informally, if a point $p = L^h_{new}(d)$ of a new wave coincides with a point $L^g_{old}(d)$ of an old wave, then the key value of p is the wave number, g, of the old wave. The other possibility is that p is an *in-between point* that does not coincide with any old point and, in this case, its key value is $g-\frac{1}{2}$, where g is the smallest wave number for which $p < L^g_{old}(d)$. Note that if wave L^{g-1}_{old} has a point on diagonal d, then clearly p lies on diagonal d between the points of the old $g-1$ and g waves; hence the term *in-between* point.

DEFINITION 3.1. *Formally, for a point p on diagonal d,*

$$key(p) = \min\ \{\ g\ :\ g \in \left\{0, \frac{1}{2}, 1, 1\frac{1}{2}, \ldots,\right\}\ \ and\ \ \left(p = L^{\lfloor g \rfloor}_{old}(d)\ \ or\ \ p < L^{\lfloor g+\frac{1}{2} \rfloor}_{old}(d)\right)\}.$$

In terms of key values, Observation 1 yields the following fact.

Observation 4. $\forall\ h, d, \quad h-1 \le key(L^h_{new}(d)) \le h+1$.

Also in terms of key values, Observations 2 and 3 yield the following.

Observation 5. If $key(L^h_{new}(d-1))$, $key(L^h_{new}(d))$, and $key(L^h_{new}(d+1))$ are all $\le$, $=$, or $\ge g$, (for $g \in \{h-1, h, h+1\}$), then $key(L^{h+1}_{new}(d))$ is $\le$, $=$, or $\ge g+1$, respectively. However, $L^{h+1}_{new}(d)$ may exist, although some of the above h-wave points do not exist, i.e., when d is in $\{-h-2, -h-1, h-1, h\}$. If those that do exist all have key values $\le g$, then it still follows that $key(L^{h+1}_{new}(d)) \le g+1$. Greater care has to be taken for the $=$ and $\ge$ case. If for all diagonals $\delta \in \{d-1, d, d+1\}$, for which $L^h_{new}(\delta)$ does not exist, $L^g_{old}(\delta)$ does not exist either, and all other relevant h-wave points have key values $=$ or $\ge g$, then $key(L^{h+1}_{new}(d))$ will still be $=$, or $\ge g+1$, respectively.

We now have the concepts and terminology needed to proceed with our central theorem.

THEOREM 3.2. *L^h_{new} is the concatenation of (up to) five pieces:* (i) *a prefix of L^{h+1}_{old},* (ii) *an in-between point p_1, with $key(p_1) = h+\frac{1}{2}$,* (iii) *a sublist of L^h_{old},* (iv) *an in-between point p_2, with $key(p_2) = h-\frac{1}{2}$, and* (v) *a suffix of L^{h-1}_{old}. Each individual piece may be empty.*

Proof. The proof of Theorem 3.2 is essentially by induction on h, but we will first show that certain monotonicity properties imply Theorem 3.2 and then prove these properties inductively.

We will prove that the following monotonicity property on key values holds for all h waves. The key values of the points along L^h_{new} are nonincreasing and strictly decreasing around in-between points as one proceeds from left to right (i.e., from diagonal $-h-1$ to $h-1$). Formally, we have the following key property.

First key property. For $d \in [-h, h-1]$, $\lfloor key(L^h_{new}(d-1)) \rfloor \ge key(L^h_{new}(d))$.

Observation 4 says that all key values on L^h_{new} are between $h-1$ and $h+1$. The *first key property* says that key values are strictly decreasing along in-between points and are otherwise nonincreasing. This implies that there is at most one point with key value $h+\frac{1}{2}$, (and it is to the right of any points with key value $h+1$, and to the left of any points with key value h), and at most one point with key value $h-\frac{1}{2}$, (to the right of any points with key value h, and to the left of any points with key value $h-1$). It follows that the *first key property* implies that L^h_{new} is the concatenation of the (up to) five pieces given in Theorem 3.2.

We shall prove by induction on h that the *first key property* holds for all h waves.

Before proceeding with a formal proof, we provide some intuition on why the Theorem holds and, at the same time, point to some of the difficulties in proving it. Suppose that L^h_{new} is indeed a concatenation of a prefix $\langle L^{h+1}_{old}(-h-1), \ldots, L^{h+1}_{old}(r)\rangle$ of L^{h+1}_{old}, a point p_1 on diagonal $r+1$, a sublist $\langle L^h_{old}(r+2), \ldots, L^h_{old}(s)\rangle$ of L^h_{old}, a point p_2 on diagonal $s+1$, and a suffix $\langle L^{h-1}_{old}(s+2), \ldots, L^h_{old}(h-1)\rangle$ of L^{h-1}_{old}. By Observation 2, clearly L^{h+1}_{new} will be equal to L^{h+2}_{old} on diagonals $-h-2 \ldots r-1$, will be equal to L^{h+1}_{old} on diagonals $r+3 \ldots s-1$, and will be equal to L^h_{old} on diagonals $s+3 \ldots h-1$. It is hence easy to see that the Theorem can be proven by induction for "most points." One of the difficulties lies in proving that at most one of the diagonals in $\{r, r+1, r+2\}$ and at most one in $\{s, s+1, s+2\}$ can contain an in-between point. A further (more serious) difficulty comes from the fact that any individual piece (of the five pieces composing L^h_{new}) can be empty, resulting in the necessity to examine an enormous number of cases. In addition, points on extreme diagonals behave differently than points on the inner diagonals. In particular, since the various scopes of the waves L^h_{new}, L^{h-1}_{new}, L^{h-1}_{old}, L^h_{old}, and L^{h+1}_{old} are different, special care has to be taken to cover all cases, where a diagonal is in the scope of one wave but outside the scope of another (see Observations 3 and 5).

Therefore, instead of proving Theorem 3.2 directly, we choose to prove that the *first key property* holds for all waves. This will allow us to reduce the number of cases we need to examine significantly, although a large number still remain.

In order to prove the *first key property*, it is very helpful to have established the following second key property for wave h, which is implied by the first key property on wave $h-1$, as shown below in Lemma 3.3.

Second key property. If $key(L^h_{new}(d)) = g + \frac{1}{2}$, then $L^{g+1}_{old}(d)$ was obtained from above.

In other words, the second key property asserts that if there is a new point p in diagonal d between the g- and $g+1$-points of the old waves, then the $g+1$-point must have been obtained from the g-point on diagonal $d+1$.

See Figure 4(a) for an illustration of the situation.

The *second key property* is quite intuitive, but its proof is not immediate. Key values (on wave $h-1$) are nonincreasing, hence if the old $g+1$ value was not reached on diagonal d in the new h wave, it would seem likely that this happened because the key value on diagonal $d+1$ of L^{h-1}_{new} "dropped below" g, and the old $g+1$ point was obtained from this "missing" old g point, $L^g_{old}(d+1)$.

The remainder of the proof of Theorem 3.2 is complex enough that we capture it in three lemmas below. First we prove in Lemma 3.3 that if the first key property holds on new wave $h-1$, then the second holds on new wave h. Then a useful corollary (Lemma 3.4) of Lemma 3.3 is given. Finally, with the aid of the second key property and its corollary, we complete the proof of Theorem 3.2 by inductively showing the first key property to hold for all waves in Lemma 3.5.

LEMMA 3.3. *If the first key property holds up to wave $h-1$ of D_{new}, then the second key property holds up to wave h of D_{new}.*

Proof. The lemma is proven by induction on h.

BASIS: Wave L^0_{new} has only one point, which is on diagonal -1. By Observation 4 $key(L^0_{new}(-1)) \leq 1$, and since L^0_{old} does not have a point on diagonal -1, we also know that $key(L^0_{new}(-1)) > 0$. Hence $key(L^0_{new}(-1))$ is either 1 or $\frac{1}{2}$. $L^1_{old}(-1)$, (whether it collides with the new zero point or not) was obtained from the (only) old zero point $L^0_{old}(0)$, i.e., from above. It follows that the *second key property* holds for wave 0 of D_{new}.

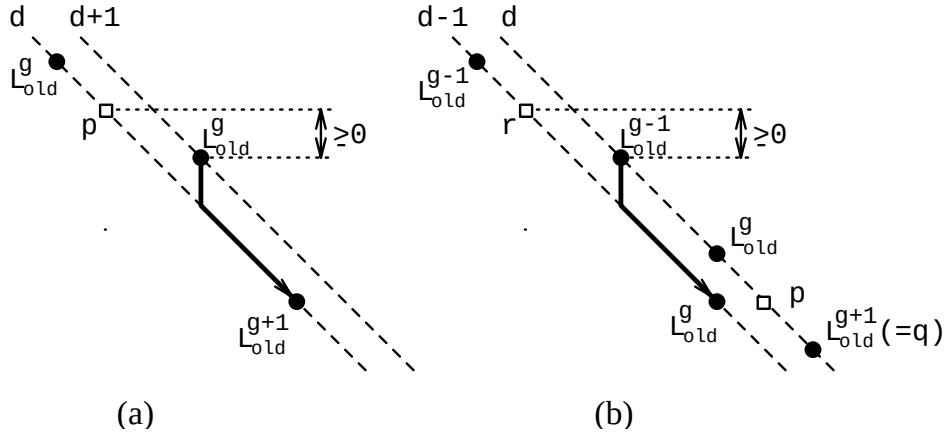


FIG. 4. *The second key property. Old points are shown as solid circles and new points as open squares.* (a) *If the new point p has key value* $g + \frac{1}{2}$, *then* $L_{old}^{g+1}(d)$ *was obtained from above, i.e., from* $L_{old}^{g}(d+1)$. (b) *Shown are new points p and r on diagonals d and* $d-1$ *with key values* $g - \frac{1}{2}$ *and* $g + \frac{1}{2}$, *respectively. The second key property implies that p could not have been obtained from r.*

INDUCTION: Assume that our claim holds for all new waves up to wave $h-1$, i.e., given an *in-between point* $L_{new}^{h-1}(d)$ with $key(L_{new}^{h-1}(d)) = g - \frac{1}{2}$, $L_{old}^{g}(d)$ was obtained from $L_{old}^{g-1}(d+1)$. As easily seen in Figure 4(a), with $p = L_{new}^{h-1}(d)$ and g replaced by $g-1$, the inductive assumption implies that

(1) if $key(L_{new}^{h-1}(d)) = g - \frac{1}{2}$, then $L_{old}^{g-1}(d+1) \geq L_{new}^{h-1}(d)$.

Henceforward, consider an in-between point $p = L_{new}^{h}(d)$, for which $key(L_{new}^{h}(d)) = g + \frac{1}{2}$. That is, $L_{old}^{g}(d)$ (if it exists) $< p < L_{old}^{g+1}(d)$, and Observation 4 further implies that $g \in \{h-1, h\}$.

To prove the *second key property* it suffices to show that $q = L_{old}^{g+1}(d)$ could not have been obtained diagonally or from the left. If $L_{old}^{g}(d)$ exists, then $Slide_d(L_{old}^{g}(d)+1)$ is at most p as $L_{old}^{g}(d) < p$ and p is a furthest point. Thus since $p < q$, q could not have been obtained diagonally.

It remains to show that q could not have been obtained from the left. Again we only need to examine the case when $L_{old}^{g}(d-1)$ exists. (If it did not exist, then clearly q could not have been obtained from the left.) This implies by Observation 3 that $d - 1 \geq -g$, and since $g \leq h$, $d-1$ is at least $-h$ and $L_{new}^{h-1}(d-1)$ exists also.

The assumption that $key(p) = g + \frac{1}{2}$, $(p = L_{new}^{h}(d))$ implies by Observation 5 that one of the three points $L_{new}^{h-1}(d-1)$, $L_{new}^{h-1}(d)$, $L_{new}^{h-1}(d+1)$ must have a key value greater than $g-1$, and since $key(L_{new}^{h-1}(d-1)) \geq key(L_{new}^{h-1}(d))$ (if it exists) $\geq key(L_{new}^{h-1}(d+1))$ (if it exists), it must always be that the key value of $r = L_{new}^{h-1}(d-1)$ is greater than $g-1$. Hence $key(r) = g - \frac{1}{2}$ or $key(r) \geq g$.

$key(r) \geq g$ means that $r = L_{new}^{h-1}(d-1) \geq L_{old}^{g}(d-1)$; hence if $q = L_{old}^{g+1}(d)$ were obtained from $L_{old}^{g}(d-1)$ (i.e., the left), then $L_{new}^{h-1}(d-1)$ would produce a point greater than or equal to q, but $L_{new}^{h}(d)$, the point reached on diagonal d, is strictly less than q, which is a contradiction.

We now show that assuming that $key(r) = g - \frac{1}{2}$ leads to a contradiction because none of the three points $L_{new}^{h-1}(d-1)$, $L_{new}^{h-1}(d)$, or $L_{new}^{h-1}(d+1)$ could have produced $L_{new}^{h}(d)$. Figure 4(b) will help in understanding the argument that follows. Since $r = L_{new}^{h-1}(d-1)$ is an in-between point in this case, it follows from (1) that $L_{old}^{g-1}(d) \geq r$; hence r cannot produce $p = L_{new}^{h}(d)$, a point with key value (much) higher than

$g-1$. On the other hand, $key(L_{new}^{h-1}(d-1)) = g - \frac{1}{2}$ implies that the key values of both $L_{new}^{h-1}(d)$ and $L_{new}^{h-1}(d+1)$ are at most $g-1$ (if they exist); hence neither could have produced $L_{new}^{h}(d)$ with key value $g + \frac{1}{2}$.

We have shown that for all legal key values of $L_{new}^{h-1}(d-1)$, q could not have been obtained from the left, and we have shown earlier that q could not have been obtained diagonally. It must be that q was obtained from the right. □

The following lemma is an immediate corollary of Lemma 3.3.

LEMMA 3.4. *If the first key property holds up to wave $h-1$, then the following holds. If $key(L_{new}^{h-1}(d)) = g - \frac{1}{2}$, then $key(L_{new}^{h}(d+1)) \leq g$. Moreover, if $L_{new}^{h}(d+1)$ was obtained from $L_{new}^{h-1}(d)$, then $key(L_{new}^{h}(d+1)) \leq g-1$.*

Proof. Since $L_{new}^{h-1}(d)$ is an in-between point, $L_{old}^{g}(d)$ was obtained from $L_{old}^{g-1}(d+1)$. As observed earlier and seen in Figure 4(a), it must be the case that $L_{old}^{g-1}(d+1) \geq L_{new}^{h-1}(d)$. Hence $Slide_{d+1}(L_{new}^{h-1}(d))$ cannot be greater than $L_{old}^{g-1}(d+1)$ (a furthest point), which proves the second part of the lemma. Moreover, if the *first key property* holds for wave $h-1$, the key values of both $L_{new}^{h-1}(d+1)$ and $L_{new}^{h-1}(d+2)$ are at most $g-1$, and hence $key(L_{new}^{h}(d+1)) \leq g$ by Observation 2. □

We conclude the proof of Theorem 3.2 by proving by induction that the *first key property* holds for all waves in D_{new}.

LEMMA 3.5. *The first key property holds for all waves in D_{new}.*

Proof. The proof is by induction on h.

BASIS: Wave 0 in D_{new} has only one point $L_{new}^{0}(-1)$. The first key property therefore trivially holds for L_{new}^{0}.

INDUCTION: Assume now that the *first key property* holds up to wave $h-1$ and the *second key property* holds up to wave h. We shall prove that the *first key property* also holds for wave h. To achieve this it suffices to examine all adjacent pairs of diagonals in wave h and to prove the required inequality on their key values. Henceforward consider the pair of points $L_{new}^{h}(d)$ and $L_{new}^{h}(d+1)$. The proof is divided into two cases depending on whether $L_{new}^{h}(d)$ is the leftmost point of the wave.

Case 1. Diagonal d is not the leftmost (lowest) diagonal on L_{new}^{h}, i.e., $d \geq -h$.

By Observation 3, L_{new}^{h-1} also has a point on diagonal d. On the other hand, we do not assume that $d < h-1$, and hence the point $L_{new}^{h-1}(d+1)$ may not exist. Let $g = \lfloor key(L_{new}^{h-1}(d)) \rfloor$ and note that by Observation 4, $g \in \{h-2, h-1, h\}$, and by definition, $key(L_{new}^{h-1}(d)) \in \{g, g+\frac{1}{2}\}$. We now further divide the proof of this case into four subcases depending on the key values of $L_{new}^{h-1}(d)$ and $L_{new}^{h-1}(d+1)$ in relation to g. Table 1 illustrates the four cases factored into two conditions on the key value of $L_{new}^{h-1}(d)$ and two conditions on the key value of $L_{new}^{h-1}(d+1)$, and for each case gives the implication on the key values of $L_{new}^{h}(d)$ and $L_{new}^{h}(d+1)$ that will be proven below. Note in each of the four cases we prove that $\lfloor key(L_{new}^{h}(d)) \rfloor \geq key(L_{new}^{h}(d+1))$.

Subcase 1(a). $key(L_{new}^{h-1}(d)) = g$ and $key(L_{new}^{h-1}(d+1)) = g$.

It suffices to show that $key(L_{new}^{h}(d)) \geq g+1 \geq key(L_{new}^{h}(d+1))$. Either $key(L_{new}^{h-1}(d+2))$ does not exist, (i.e., $d+1$ is the rightmost diagonal of L_{new}^{h-1}), or by the *first key property* on wave $h-1$ $key(L_{new}^{h-1}(d+2)) \leq g$. In either case, Observation 5 implies $key(L_{new}^{h}(d+1)) \leq g+1$. We now show that $key(L_{new}^{h}(d)) \geq g+1$. Either $key(L_{new}^{h-1}(d-1)) \geq g$, (by the *first key property* on wave $h-1$), or $key(L_{new}^{h-1}(d-1))$ does not exist because $d = -h$ is the leftmost diagonal of wave L_{new}^{h-1}, in which case, by Observation 3, neither $L_{old}^{h-2}, L_{old}^{h-1}$, nor L_{old}^{h} have points on diagonal $d-1$, and d is also the leftmost diagonal on L_{old}^{g}. In either case, Observation 5 implies $key(L_{new}^{h}(d)) \geq g+1$.

Subcase 1(b). $key(L_{new}^{h-1}(d)) = g$ and $key(L_{new}^{h-1}(d+1)) \leq g - \frac{1}{2}$ (or doesn't exist).

If $key(L_{new}^{h}(d+1)) \leq g$ we are immediately done, as $L_{new}^{h}(d)$ is always greater than $L_{new}^{h-1}(d)$, and since $key(L_{new}^{h-1}(d)) = g$, we have $key(L_{new}^{h}(d)) > g$. So

TABLE 1

The key values of the pair $L^h_{new}(d), L^h_{new}(d+1)$, as implied by the key values of the pair $L^{h-1}_{new}(d), L^{h-1}_{new}(d+1)$.

	$key(L^{h-1}_{new}(d+1)) = g$	$key(L^{h-1}_{new}(d+1)) \leq g - \frac{1}{2}$ (or doesn't exist)
$key(L^{h-1}_{new}(d)) = g$	$key(L^h_{new}(d)) \geq g+1 \geq key(L^h_{new}(d+1))$	$key(L^h_{new}(d)) > g \geq key(L^h_{new}(d+1))$ or $key(L^h_{new}(d)) \geq g+1 \geq key(L^h_{new}(d+1)) > g$
$key(L^{h-1}_{new}(d)) = g + \frac{1}{2}$	$key(L^h_{new}(d)) \geq g+1 \geq key(L^h_{new}(d+1))$	$key(L^h_{new}(d)) \geq g + \frac{1}{2} > key(L^h_{new}(d+1))$

we assume $key(L^h_{new}(d+1)) \geq g + \frac{1}{2}$ and show that $key(L^h_{new}(d)) \geq g + 1 \geq key(L^h_{new}(d+1))$. $L^h_{new}(d+1)$ got its value from one of the points in $\{L^{h-1}_{new}(d), L^{h-1}_{new}(d+1), L^{h-1}_{new}(d+2)\}$. Since all three points have key values of g or less, clearly $key(L^h_{new}(d+1)) \leq g+1$. In addition, since $key(L^{h-1}_{new}(d+1))$ is either $g - \frac{1}{2}$ or $\leq g-1$, $key(L^{h-1}_{new}(d+2))$ (if it exists) $\leq g-1$ by the first key property, and cannot produce $L^h_{new}(d+1)$, a point with key value greater than g. $L^{h-1}_{new}(d+1)$ (if it exists) $< L^g_{old}(d+1)$, (by definition of Subcase 1(b)) and hence $L^{h-1}_{new}(d+1)$ cannot produce a point beyond $L^g_{old}(d+1)$ either. It follows that $L^h_{new}(d+1)$ got its value from the left, (i.e., from $L^{h-1}_{new}(d) = L^g_{old}(d)$). Since $key(L^h_{new}(d+1))$ is also $\geq g + \frac{1}{2}$, it must be the case that

$$L^{h-1}_{new}(d) > L^g_{old}(d+1); \text{ and by Subcase 1(b) } L^g_{old}(d+1) > L^{h-1}_{new}(d+1).$$

It follows that $L^h_{new}(d)$ was not obtained from above. If $L^{h-1}_{new}(d-1)$ exists, then $key(L^{h-1}_{new}(d-1)) \geq g$, (by the first key property), otherwise $d = -h$ and $L^g_{old}(d-1)$ does not exist either (since by Observation 3 none of the three relevant old waves have a point on diagonal $-h$). Thus $L^h_{new}(d)$ was obtained from the left or diagonally, and both of these source points in the L^{h-1}_{new} wave have key values of g or greater (or do not exist on both L^{h-1}_{new} and L^g_{old}). Thus by Observation 5 $key(L^h_{new}(d)) \geq g+1$.

Subcase 1(c). $key(L^{h-1}_{new}(d)) = g + \frac{1}{2}$ and $key(L^{h-1}_{new}(d+1)) = g$.
Immediately note that because $key(L^{h-1}_{new}(d)) = g + \frac{1}{2} \in [h-2, h]$ by Observation 4, it follows that $g < h$. It suffices to show that $key(L^h_{new}(d)) \geq g+1 \geq key(L^h_{new}(d+1))$. Lemma 3.4 immediately implies that $key(L^h_{new}(d+1)) \leq g+1$. $L^{h-1}_{new}(d)$ and $L^{h-1}_{new}(d+1)$ have key values g or more and the key value of $L^{h-1}_{new}(d-1)$ (if it exists) is also $> g$ (by the first key property). If $L^{h-1}_{new}(d-1)$ does not exist, then $L^g_{old}(d)$ does not exist either, (Observation 3), and it follows from Observation 5 that $key(L^h_{new}(d)) \geq g+1$.

Subcase 1(d). $key(L^{h-1}_{new}(d)) = g + \frac{1}{2}$ and $key(L^{h-1}_{new}(d+1)) \leq g - \frac{1}{2}$ (or doesn't exist).
It suffices to show that $key(L^h_{new}(d)) \geq g + \frac{1}{2} > key(L^h_{new}(d+1))$. Clearly $L^h_{new}(d) > L^{h-1}_{new}(d)$ and hence $key(L^h_{new}(d)) \geq g + \frac{1}{2}$. The first key property on wave $h-1$ implies that $key(L^{h-1}_{new}(d+2)) \leq g-1$ (if it exists); hence $key(L^h_{new}(d+1)) \leq g$ if it was obtained from above. Similarly, $key(L^h_{new}(d+1)) \leq g$ if it was obtained diagonally, as $key(L^{h-1}_{new}(d+1)) \leq g - \frac{1}{2}$. Lastly, the second part of Lemma 3.4 also implies $key(L^h_{new}(d+1)) \leq g$ if it was obtained from $L^{h-1}_{new}(d)$, whose key value is $g + \frac{1}{2}$. Thus, regardless of how it was obtained, $key(L^h_{new}(d+1)) \leq g$.

Case 2. Diagonal d is the leftmost (lowest) diagonal on L^h_{new}, i.e., $d = -h-1$. This case is not covered by Table 1, since $L^{h-1}_{new}(d)$ does not exist. We need

to prove that $\lfloor key(L^h_{new}(-h-1))\rfloor \geq key(L^h_{new}(-h))$. The key value of the point $L^{h-1}_{new}(-h)$, (the leftmost point on L^{h-1}_{new}), is strictly greater than $h-1$, since L^{h-2}_{old} and L^{h-1}_{old} do not have a point on diagonal $-h$, (Observation 3), and obviously $\leq h$ by Observation 4; hence $key(L^{h-1}_{new}(-h)) \in \{h, h-\frac{1}{2}\}$.

Subcase 2(a). $key(L^{h-1}_{new}(-h)) = h$.

L^{h-1}_{new} does not have a point on either diagonal $-h-1$ or $-h-2$; hence $L^h_{new}(-h-1)$ was obtained from $L^{h-1}_{new}(-h)$ which is the same point as $L^h_{old}(-h)$ since $key(L^{h-1}_{new}(-h)) = h$. L^h_{old} does not have points on these two diagonals either; hence $L^{h+1}_{old}(-h-1)$ was obtained from $L^h_{old}(-h)$. It follows that $L^h_{new}(-h-1) = L^{h+1}_{old}(-h-1)$, or equivalently $key(L^h_{new}(-h-1)) = h+1$. Since all key values on the new h wave are less or equal to $h+1$, (Observation 4), $key(L^h_{new}(-h)) \leq h+1$, which proves the required inequality.

Subcase 2(b). $key(L^{h-1}_{new}(-h)) = h-\frac{1}{2}$.

In this case $key(L^{h-1}_{new}(-h+1))$(if it exists) $\leq h-1$, (by the first key property), (while $L^{h-1}_{new}(-h-1)$ does not exist), and hence if $L^h_{new}(-h)$ was obtained from above, then $key(L^h_{new}(-h)) \leq h$. On the other hand, if $L^h_{new}(-h)$ was obtained diagonally, we also have $key(L^h_{new}(-h)) \leq h$, since $L^{h-1}_{new}(-h) < L^h_{old}(-h)$, and $L^h_{old}(-h)$ is a furthest point. On the other hand, (as noted earlier), the key value of the lowest diagonal on the new h wave ($L^h_{new}(-h-1)$) is always strictly greater than h, which proves that $key(L^h_{new}(-h-1)) > h \geq key(L^h_{new}(-h))$. This terminates the proof of this last subcase and hence the proof of Lemma 3.5. □

In summary, Observation 4 says that all key values on L^h_{new} are between $h-1$ and $h+1$, and Lemma 3.5 shows that for all diagonals $-h-1 \leq d < h-1$, $\lfloor key(L^h_{new}(d))\rfloor \geq key(L^h_{new}(d+1))$. The key values are hence nondecreasing and L^h_{new} has at most two in-between points. It follows that Theorem 3.2 holds: L^h_{new} can be pasted together from a prefix of L^{h+1}_{old}, a sublist of L^h_{old}, and a suffix of L^{h-1}_{old} and at most two additional points, which may occur between sublists. Any individual piece may be empty. □

3.1. Formal presentation of algorithm. We now show how Theorem 3.2 and the lemmas and observations of the previous subsection directly lead to an efficient algorithm for computing $L^0_{new} \ldots L^k_{new}$ from $L^0_{old} \ldots L^k_{old}$.

Suppose the wave L^h_{new} has been constructed, and let $L^h_{new}[a \ldots b]$ denote the sublist $\langle L^h_{new}(a), L^h_{new}(a+1), \ldots, L^h_{new}(b)\rangle$ of L^h_{new}, (when $b < a$ $L^h_{new}[a \ldots b]$ denotes the empty list). Let $\delta_1 \in \{0,1\}$ be the number of points on L^h_{new} with key value $h+\frac{1}{2}$, and let $\delta_2 \in \{0,1\}$ be the number of points with key value $h-\frac{1}{2}$. Define p^h_1 to be the largest (rightmost) diagonal for which $key(L^h_{new}(p^h_1)) = h+1$, or let p^h_1 be $-h-2$ if no such diagonal exists. Similarly, let $p^h_2 = \max\{d : (\, key(L^h_{new}(d)) = h\,)$ or $(\, d = p^h_1 + \delta_1\,)\}$. Theorem 3.2 says that $L^h_{new}[-h-1 \ldots p^h_1] = L^{h+1}_{old}[-h-1 \ldots p^h_1]$, $L^h_{new}[p^h_1 + 1 + \delta_1 \ldots p^h_2] = L^h_{old}[p^h_1 + 1 + \delta_1 \ldots p^h_2]$, and that $L^h_{new}[p^h_2 + 1 + \delta_2 \ldots h-1] = L^{h-1}_{old}[p^h_2 + 1 + \delta_2 \ldots h-1]$. Notice that the above equalities hold even if individual pieces are empty. By Observation 2 it follows that if $L^h_{new}[a \ldots b] = L^g_{old}[a \ldots b]$, then $L^{h+1}_{new}[a+1 \ldots b-1] = L^{g+1}_{old}[a+1 \ldots b-1]$. In addition, if $g = h+1$, we also have $L^{h+1}_{new}[a] = L^{g+1}_{old}[a]$, and if $g = h-1$ we also have $L^{h+1}_{new}[b] = L^{g+1}_{old}[b]$. Thus, all of L^{h+1}_{new} is determined by these equalities except for the two or three points on the diagonals in the interval $[p^h_1, p^h_1 + 1 + \delta_1]$ and the two or three points on the diagonals in the interval $[p^h_2, p^h_2 + 1 + \delta_2]$. In addition, Lemma 3.4 implies that if $key(L^h_{new}(p^h_2 + 1)) = h - \frac{1}{2}$, (i.e., $\delta_2 = 1$), then $key(L^{h+1}_{new}(p^h_2 + 2)) \leq h$, and hence $key(L^{h+1}_{new}(p^h_2 + 2)) = h$ (since all key values on L^{h+1}_{new} are at least h). Simi-

larly, if $key(L^h_{new}(p^h_1+1)) = h+\frac{1}{2}$, (i.e., $\delta_1 = 1$), then $key(L^{h+1}_{new}(p^h_1+2)) \leq h+1$, and if in addition $p^h_2 > p^h_1+2$, then $key(L^{h+1}_{new}(p^h_1+2)) = h+1$. Thus, the only points on L^{h+1}_{new} that cannot be determined by merely examining the key value of the points on L^h_{new} are the four points on diagonals p^h_1, p^h_1+1, p^h_2, and p^h_2+1.

If L^g_{old} is a doubly linked list so that, given a pointer to $L^g_{old}(d)$, one can move to $L^g_{old}(d-1)$ or $L^g_{old}(d+1)$ in constant time, and each diagonal across the waves is a doubly linked list, so that given a pointer to $L^g_{old}(d)$, one can move to $L^{g+1}_{old}(d)$ or $L^{g-1}_{old}(d)$ in constant time, we can construct L^{h+1}_{new} from L^h_{new} and the old L waves by performing at most $O(1)$ computations and pointer changes. To do so requires knowing the *break diagonals* p^h_1 and p^h_2 of L^h_{new} and determining the new *break diagonals* p^{h+1}_1 and p^{h+1}_2 during the computation of L^{h+1}_{new}. Notice that all but $O(1)$ of the diagonal links from L^h_{new} to L^{h+1}_{new} will be imported directly from the corresponding diagonal links of L^{h-1}_{old}, L^h_{old}, and L^{h+1}_{old} so that only those for the points on diagonals p^h_1, p^h_1+1, p^h_2, p^h_2+1 need to be recomputed. A formal algorithmic description is given in Figures 5, 6, and 7, which shows that such a cross-linked structure can be maintained to realize an incremental update in $O(k)$ time of $L^0_{new}, \ldots, L^{k-1}_{new}$.

For reasons of simplicity, the explicit algorithm in Figures 5, 6, and 7 is given as if the waves were stored in an array and $O(1)$ access to $L^h_{new}(d)$, for any d, were possible. The cross-linked structure is clearly necessary to realize an incremental update in $O(k)$ time, but the description of the algorithm is much simpler in the latter form. The reader can verify that, given the cross-linked structure and pointers to $L^h_{new}(p^h_1)$ and $L^h_{new}(p^h_2)$, as well as to $L^{h+1}_{old}(p^h_1+1)$ and $L^h_{old}(p^h_2+1)$, every pertinent wave element in *Construct_new_wave* in Figure 6 is $O(1)$ links away from either of these four "break pointers," or the first element of a completed new wave or an as yet unused old wave. In more detail (which is not necessary to understand the explicit algorithms in Figures 5, 6, and 7), the following elements can be accessed in $O(1)$ time.

1. The first $O(1)$ elements of L^h_{new} and L^{h+2}_{old}: $L^h_{new}(-h-1+O(1))$ and $L^{h+2}_{old}(-h-2+O(1))$.
2. All points on L^h_{new} within $O(1)$ diagonals of p^h_1 and p^h_2: $L^h_{new}(p^h_1 \pm O(1))$ and $L^h_{new}(p^h_2 \pm O(1))$.
3. Points on old waves that are identical to the above points: $L^{h+1}_{old}[-h-1 \ldots p^h_1] = L^h_{new}[-h-1 \ldots p^h_1]$.
4. The first $O(1)$ points on the suffixes of the relevant old L lists: $L^{h+1}_{old}(p^h_1+1+O(1))$ and $L^h_{old}(p^h_2+1+O(1))$.
5. Points accessible from any of the above points through old diagonal links: $L^{h+2}_{old}(p^h_1)$, which is diagonally linked to $L^{h+1}_{old}(p^h_1) = L^h_{new}(p^h_1)$, as well as $L^{h+2}_{old}(p^h_1 \pm O(1))$. If $L^h_{new}(p^h_2) = L^h_{old}(p^h_2)$ (which is always true when $p^h_2 > p^h_1+1$), then $L^{h+1}_{old}(p^h_2)$ is accessible through the diagonal link from $L^h_{old}(p^h_2)$, and hence $L^{h+1}_{old}(p^h_2 \pm O(1))$ is also accessible in $O(1)$ time. If $L^h_{new}(p^h_2) < L^h_{old}(p^h_2)$, (and hence $p^h_2 \in \{p^h_1, p^h_1+1\}$), then $L^{h+1}_{old}(p^h_2 \pm O(1))$ is still accessible in $O(1)$ steps, as $p^h_2 = p^h_1 + O(1)$.

The computation of the last wave L^k_{new} and its diagonal links from and to L^{k-1}_{new} may take an additional $O(k)$ time since the required L^{k+1}_{old} piece and its diagonal links from L^k_{old} have not been computed previously and need to be computed now (see lines 6 and 7 in Figure 7). In total, the computation of the k waves requires computing $O(k)$ new points and updating $O(k)$ links. The computation of each new point is done by calling the *Slide* function (line 2 in Figure 6), which takes $O(1)$ time. In total the computation of the k waves for D_{new} takes $O(k)$ time, when given the k waves for D, cross-linked across waves and diagonals as described above.

Procedure Construct_new_wave$(h, p_1^h, p_2^h, p_1^{h+1}, p_2^{h+1})$
1. $L_{new}^{h+1}[-h-2 \ldots p_1^h-1] \leftarrow L_{old}^{h+2}[-h-2 \ldots p_1^h-1]$
2. Compute($L_{new}^{h+1}(p_1^h)$)
3. Compute($L_{new}^{h+1}(p_1^h+1)$)
4. $p_1^{h+1} \leftarrow \begin{cases} p_1^h - 1 & \text{if } L_{new}^{h+1}(p_1^h) < L_{old}^{h+2}(p_1^h) \\ p_1^h & \text{if } L_{new}^{h+1}(p_1^h) = L_{old}^{h+2}(p_1^h) \text{ and } L_{new}^{h+1}(p_1^h+1) < L_{old}^{h+2}(p_1^h+1) \\ p_1^h + 1 & \text{otherwise} \end{cases}$
5. **if** $p_2^h > p_1^h + 2$ **then**
6. { $L_{new}^{h+1}[p_1^h+2 \ldots p_2^h-1] \leftarrow L_{old}^{h+1}[p_1^h+2 \ldots p_2^h-1]$
7. Double_Link($L_{new}^{h+1}(p_1^h+2)$, $L_{new}^{h+1}(p_1^h+1)$)
8. }
9. **if** $p_2^h \geq p_1^h + 2$ **then** Compute($L_{new}^{h+1}(p_2^h)$)
10. **if** $p_2^h \geq p_1^h + 1$ **then** Compute($L_{new}^{h+1}(p_2^h+1)$)
11. $p_2^{h+1} \leftarrow \begin{cases} p_2^h - 1 & \text{if } L_{new}^{h+1}(p_2^h) < L_{old}^{h+1}(p_2^h) \\ p_2^h & \text{if } L_{new}^{h+1}(p_2^h) = L_{old}^{h+1}(p_2^h) \text{ and } L_{new}^{h+1}(p_2^h+1) < L_{old}^{h+1}(p_2^h+1) \\ p_2^h + 1 & \text{otherwise} \end{cases}$
12. **if** $p_2^h + 2 \leq h$ **then**
13. { $L_{new}^{h+1}[p_2^h+2 \ldots h] \leftarrow L_{old}^{h}[p_2^h+2 \ldots h]$
14. Double_Link($L_{new}^{h+1}(p_2^h+2)$, $L_{new}^{h+1}(p_2^h+1)$)
15. }

FIG. 5. *Construction of L_{new}^{h+1} from L_{new}^{h} and auxiliary pointers.*

Procedure Compute$(L_x^h(d))$
1. **if** $x = new$ **then** $\delta \leftarrow 1$ **else** $\delta \leftarrow 0$
2. $L_x^h(d) \leftarrow Slide_d \left(\max \begin{cases} L_x^{h-1}(d+1)+1 & \text{if } d < h-1-\delta \\ L_x^{h-1}(d)+1 & \text{if } -h-\delta < d < h-\delta \\ L_x^{h-1}(d-1) & \text{if } d > -h+1-\delta \end{cases} \right)$
3. **if** $h > 0$ **and** $d > -h-\delta$ **then**
4. { Double_Link($L_x^h(d)$, $L_x^h(d-1)$)
5. **if** $d < h-\delta$ **then** Double_Link($L_x^h(d)$, $L_x^{h-1}(d)$)
6. }

FIG. 6. *Computation of $L_{new}^h(d)$ or $L_{old}^h(d)$ and update of all relevant pointers.*

Procedure New_Wave
1. $L_{new}^0(-1) \leftarrow Slide_{-1}(0)$
2. $p_2^0 \leftarrow -1$
3. **if** $L_{new}^0(-1) < L_{old}^1(-1)$ **then** $p_1^0 \leftarrow -2$ **else** $p_1^0 \leftarrow -1$
4. **for** $h \leftarrow 0$ **to** $k-2$ **do**
5. Construct_new_wave$(h, p_1^h, p_2^h, p_1^{h+1}, p_2^{h+1})$
6. **for** $d \leftarrow -k-1$ **to** p_1^k+1 **do**
7. Compute($L_{old}^{k+1}(d)$)
8. Construct_new_wave$(k-1, p_1^{k-1}, p_2^{k-1}, p_1^k, p_2^k)$

FIG. 7. *Construction of $L_{new}^0 \ldots L_{new}^k$ from the corresponding old waves.*

3.2. An analogous theorem for longest common subsequences. As pointed out in the introduction, the model in which one allows insertions and deletions, called indels, but not mismatches is an important variation because it is dual

to finding the longest common subsequence. A theorem similar to Theorem 3.2, but simpler, holds in this situation. We simply sketch the result here, since the proof method and outline remain the same.

When mismatches are not allowed, several of the preliminary lemmas of section 2 change in small ways. In Lemma 2.1, the term δ_{a_i,b_j} should be multiplied by two, as a substitution can now only be achieved by an insertion and deletion. In Lemma 2.3, values increase along a diagonal by zero or two, i.e., $D[i,j]-D[i–1,j–1] \in \{0,2\}$. Next note that with an even number of indels one must end in an even numbered diagonal and an odd diagonal with an odd number of indels. Thus for this variation of the problem, an h-wave has $h+1$ points on every other diagonal, i.e., $L^h = \langle L^h(-h), L^h(-h+2), \ldots, L^h(h-2), L^h(h)\rangle$. Finally, in Lemma 2.8, one must drop the term $L^{h-1}(d)+1$ from the 3-way maximum as it represents the contribution of extension via a substitution. As a consequence, a point is either obtained from the *left* or from *above*.

We continue to define key values as in Definition 3.1. Note however that the diagonals on a new h wave do not contain old h points, and hence an in-between point p on L^h_{new} is between the old $h-1$ and the old $h+1$ wave. As a result of our definitions, a key value of a point p on a new h wave will assume one of the three values in $\{h-1, h+\frac{1}{2}, h+1\}$. In spirit, all the observations in section 3 continue to hold with the obvious modifications that follow from the fact that points can only be obtained in two ways. Theorem 3.2 becomes the following.

THEOREM 3.6. *Wave h in D_{new} (L^h_{new}) is the concatenation of (up to) three pieces:* (i) *a prefix of L^{h+1}_{old},* (ii) *an in-between point p with key$(p) = h+\frac{1}{2}$, and* (iii) *a suffix of L^{h-1}_{old}. Not all pieces must be present.*

The proof of Theorem 3.6 is in some sense a subset of the proof of Theorem 3.2. So rather than prove it formally, we sketch the main elements and leave the details as an exercise. The informal statement of the first key property remains the same, but its formal statement becomes the following.

First key property. For $d \in [-h+1, h-1]$, $\lfloor key(L^h_{new}(d-2))\rfloor \geq key(L^h_{new}(d))$, to account for the fact that only alternate diagonals are relevant. This first key property can similarly be proven by induction on h using the second key property and its immediate consequence, the analogue of Lemma 3.4.

Second key property. If $key(L^h_{new}(d)) = h+\frac{1}{2}$, then $L^{h+1}_{old}(d)$ was obtained from above.

Because the proof in this case is simpler, we combine the proof of the second key property and of the analogue of Lemma 3.4 in Lemma 3.7.

LEMMA 3.7. *If the first key property holds up to wave h, and if key$(L^h_{new}(d)) = h+\frac{1}{2}$, then* (1) *key$(L^{h+1}_{old}(d))$ was obtained from above, and key$(L^{h+1}_{new}(d+1)) = h$.*

Proof (sketch). The proof is by induction.

BASIS: We need to show that if $key(L^0_{new}(-1)) = \frac{1}{2}$, then $L^1_{old}(-1)$ was obtained from above and $key(L^1_{new}(0)) = 0$. $L^1_{old}(-1)$ (whether it collides with $L^0_{new}(-1)$ or not) was obtained from the (only) old zero point $L^0_{old}(0)$, i.e., from above. In addition, if $key(L^0_{new}(-1)) = \frac{1}{2}$, then $L^0_{new}(-1)$ must be $\leq L^0_{old}(0)$; hence $L^1_{new}(0)$ will be equal to $L^0_{old}(0)$.

INDUCTION: Assume that Lemma 3.7 holds up to wave h. Consider an in-between point $L^h_{new}(d)$, for which $key(L^h_{new}(d)) = h+\frac{1}{2}$. We need to show that $L^{h+1}_{old}(d)$ could not have been obtained from the left and that $key(L^{h+1}_{new}(d+1)) = h$. $key(L^h_{new}(d)) = h+\frac{1}{2}$ implies that one of the two points $L^{h-1}_{new}(d-1)$, $L^{h-1}_{new}(d+1)$ must have a key value of $h-\frac{1}{2}$ or h, and since $key(L^{h-1}_{new}(d-1)) \geq key(L^{h-1}_{new}(d+1))$, it must be that $key(L^{h-1}_{new}(d-1)) \in \{h-\frac{1}{2}, h\}$. $key(L^{h-1}_{new}(d-1)) = h-\frac{1}{2}$ can be ruled out, since it

Procedure Construct_new_LCS_wave(h, p^h, p^{h+1})
1. $L^{h+1}_{new}[-h-2...p^h-1] \leftarrow L^{h+2}_{old}[-h-2...p^h-1]$
2. Compute_LCS($L^{h+1}_{new}(p^h+1)$)
3. $p^{h+1} \leftarrow \begin{cases} p^h-1 & \text{if } L^{h+1}_{new}(p^h+1) < L^{h+2}_{old}(p^h+1) \\ p^h+1 & \text{otherwise} \end{cases}$
4. $L^{h+1}_{new}[p^h+3..h] \leftarrow L^h_{old}[p^h+3...h]$
5. Double_Link($L^{h+1}_{new}(p^h+3)$, $L^{h+1}_{new}(p^h+1)$)

FIG. 8. *Construction of L^{h+1}_{new} from L^h_{new} and auxiliary pointers.*

Procedure Compute_LCS($L^h_x(d)$)
1. **if** $x = new$ **then** $\delta \leftarrow 1$ **else** $\delta \leftarrow 0$
2. $L^h_x(d) \leftarrow Slide_d\left(\max \left\{ \begin{array}{ll} L^{h-1}_x(d+1)+1 & \text{if } d < h-\delta \\ L^{h-1}_x(d-1) & \text{if } d > -h-\delta \end{array} \right\} \right)$
3. **if** $h > 0$ **and** $d > -h-\delta$ **then**
4. { Double_Link($L^h_x(d)$, $L^h_x(d-2)$)
5. **if** $d < h-\delta$ **then** Double_Link($L^h_x(d)$, $L^{h-1}_x(d+1)$)
6. }

FIG. 9. *Computation of $L^h_{new}(d)$ or $L^h_{old}(d)$ and update of all relevant pointers.*

contradicts the inductive hypotheses, which says that in this case $key(L^h_{new}(d)) = h-1$. If $L^{h+1}_{old}(d)$ was obtained from $L^{h-1}_{new}(d-1)$ (i.e., the left) and $key(L^{h-1}_{new}(d-1)) = h$, we would have $key(L^h_{new}(d)) = h+1$; again a contradiction. Hence $L^{h+1}_{old}(d)$ was obtained from $L^h_{old}(d+1)$ (i.e., from above). This in turn implies that $L^h_{old}(d+1) \geq L^h_{new}(d)$. If $L^{h+1}_{new}(d+1)$ was obtained from $L^h_{new}(d)$, then the above inequality would imply that $L^{h+1}_{new}(d+1) \leq L^h_{old}(d+1)$, but since $L^{h+1}_{new}(d+1) \geq L^h_{old}(d+1)$ (always), it follows in this case that $L^{h+1}_{new}(d+1) = L^h_{old}(d+1)$. On the other hand by the first key property on wave h, $key(L^h_{new}(d+2)) = h-1$; hence if $L^{h+1}_{new}(d+1)$ was obtained from $L^h_{new}(d+2)$, we also have $L^{h+1}_{new}(d+1) = L^h_{old}(d+1)$. Hence regardless of how $L^{h+1}_{new}(d+1)$ was obtained, $key(L^{h+1}_{new}(d+1)) = h$. □

Given the second key property and Lemma 3.7, the proof of the first key property, and hence Theorem 3.6, is outlined in a nutshell as follows. Given a pair of consecutive points on wave h, $L^h_{new}(d-2)$ and $L^h_{new}(d)$, we consider the point $L^{h-1}_{new}(d-1)$. If $key(L^{h-1}_{new}(d-1)) = g$, $(g \in \{h-2, h\})$, then $key(L^h_{new}(d-2)) \geq g+1$, while $key(L^h_{new}(d)) \leq g+1$. If $L^{h-1}_{new}(d-1)$ is an in-between point, then $key(L^{h-1}_{new}(d-1)) = h-\frac{1}{2}$, and by Lemma 3.7 $key(L^h_{new}(d)) = h-1$, while $key(L^h_{new}(d-2))$ is always greater than or equal to $h-1$.

3.3. Explicit algorithm for LCS. The explicit algorithm given in Figures 8, 9, and 10 is similar to and simpler than the one for edit distance. Waves are still doubly linked lists, with $L^h(d)$ doubly linked to $L^h(d-2)$ and $L^h(d+2)$. Diagonal-links will be slightly different in that $L^h(d+1)$ is linked to $L^{h+1}(d)$ (as L^{h+1} does not have a point on diagonal $d+1$). p^h is defined as the largest (rightmost) diagonal on L^h_{new} for which $key(L^h_{new}(p^h)) = h+1$ and is set to $-h-2$ if there is no such diagonal.

4. Four applications. The power of the incremental algorithm of the proceeding section is that it delivers an encoding of the dynamic programming solution for each and every problem so obtained. In the context of the four applications of this section, this feature of the method allows the algorithm to completely explore the space of solutions to each subproblem. In the case of some of the applications, this

Procedure New_LCS_Wave
1. $L^0_{new}(-1) \leftarrow Slide_{-1}(0)$
2. **if** $L^0_{new}(-1) < L^1_{old}(-1)$ **then** $p^0 \leftarrow -2$ **else** $p^0 \leftarrow -1$
3. **for** $h \leftarrow 0$ **to** $k-2$ **do**
4. Construct_new_LCS_wave(h, p^h, p^{h+1})
5. **for** $d \leftarrow -k-1$ **to** $p^k_1 + 1$ **by** 2 **do**
6. Compute_LCS($L^{k+1}_{old}(d)$)
7. Construct_new_LCS_wave$(k-1, p^{k-1}, p^k)$

FIG. 10. *Construction of $L^0_{new} \dots L^k_{new}$ from the corresponding old waves.*

is essential to their efficient solution, and in others it provides leverage not found in previous algorithms that can only keep track of the best solution to a set of subproblems. For each application, we show how to apply the incremental algorithm as a subprocedure and focus on how its wave structure is processed at each stage to provide the desired solutions.

Before proceeding with the description of the applications, we introduce a notation for the wave structure that is different from the one introduced in section 3 but is convenient for the discussion of applications. In section 3 we showed how to compute the $k+1$ waves $L^0_{new}, \dots, L^k_{new}$ of the edit-distance matrix $D_{new}[i,j]$ of A and bB, when given the $k+1$ *waves* $L^0_{old}, \dots, L^k_{old}$ of the matrix $D_{old}[i,j]$ of A versus B. In all of our applications the overall outline of the algorithm is to start with a k-thresholded solution for A versus some suffix B_s of B. (Recall that B^r_l is the substring $b_{l+1}b_{l+2}\dots b_r$, and that $B_l = B^n_l$ and $B^r = B^r_0$.) When $s = n$ this is the trivial solution of A versus the empty string, and when $s < n$ the initial solution can be computed with the standard $O(n + k^2)$ algorithm. Thereafter, the application algorithm incrementally computes the solutions to A versus B_l for $l = s-1, s-2, \dots, 0$ using the incremental algorithm of section 3 as a subroutine.

At any moment we will have $k+1$ waves L^0, L^1 ... L^k modeling the solution to a comparison between some suffix B_l of B and A. Let the *origin diagonal* for this problem be l, and denote by D_l the dynamic programming matrix of the comparison of A with B_l. Further let $C^h(d) = L^h(l+d)$ denote the wave value that is at diagonal d of the matrix D_l. In this way $L^h = \langle C^h(-h), C^h(-(h-1)), \dots, C^h(h-1), C^h(h)\rangle$ regardless of the suffix of B at hand. Recall from the preliminaries that, in order to be compliant with the recursion for computing L^h in terms of L^{h-1}, it was convenient to sometimes set $L^h(d)$ to ∞ even though the extreme point on diagonal d in D had value h. But in the context of our applications, it is now convenient to reset these ∞ values so that C^h always holds the furthest h point on the corresponding diagonal, *within the boundary of* D, if such an h point exists. Formally if $L^h(l+d) = \infty$ while $L^{h-1}(l+d) \neq p_{last}(d)$, where $p_{last}(d) = \min\{m, n-l-d\}$ is the row number of the last point on diagonal d in matrix D, then set $C^h(d) = p_{last}(d)$. Note that the D value of $p_{last}(d)$ on diagonal d is indeed equal to h in this case. It is easy to see that this "adjustment" of values can be done $O(k)$ time per D_l matrix.

Recall from the previous section that each wave is implemented as a doubly linked list so that given a pointer to $C^h(d)$ one can move to $C^h(d-1)$ or $C^h(d+1)$ in constant time, and each diagonal across the waves is in a doubly linked list so that given a pointer to $C^h(d)$ one can move to $C^{h+1}(d)$ or $C^{h-1}(d)$ in constant time.

4.1. Approximate string matching and longest prefix. The approximate string matching problem was used in the introduction as an example of how our incremental alignments algorithm could be used to find *all* matching substrings in $O(nk)$ time. We begin by exploring in greater detail how our incremental algorithm

is applied to the following slightly more general problem and the leverage it provides over other algorithms. Consider first finding for each position $l \in [0, n]$ of a text B the length $m(l)$ of the longest prefix $A^{m(l)}$ of A that can be matched to some prefix of B_l with no more than k differences. Formally, $m(l) = \max\{p \in [0, m] : \exists r \in [l, n],\ ED(A^p, B_l^r) \leq k\}$. Further consider finding for each l the set of all r such that the substring B_l^r of the text B and the prefix $A^{m(l)}$ are within edit distance k of each other. When $m(l) = m$ we call such a substring of the text a *k-match*, and otherwise a *longest prefix k-match*. In summary, the *longest prefix approximate match problem* is, given strings A, B, and threshold k, to find for each l the length $m(l)$ and the set of indices r such that $ED(A^{m(l)}, B_l^r) \leq k$. The problem obviously generalizes the approximate string matching problem which seeks all (l, r) for which $m(l) = m$.

We start by building the trivial k-thresholded solution for A versus B_n $(= \varepsilon)$ and then proceed incrementally. First observe that constructing the trivial solution for A versus B_n simply requires building an initial cross-linked wave structure and setting $C^h(d)$ to h if $d = -h$ and to ∞ otherwise. The algorithm then proceeds to produce solutions for A versus progressively longer suffixes of B taking $O(k)$ time per incremental shift using our central result. The only remaining detail is to show how, given the k-thresholded solution for A versus some suffix B_l, one finds the longest prefix of A and all r and h such that $ED(A^{m(l)}, B_l^r) = h \leq k$. Note that when $m(l) < m$ the corresponding edit distance is always k, since we would otherwise choose a longer prefix of A. Even when $m(l) = m$ there is at least one diagonal d for which $C^k(d)$ is exactly m. This is easily seen by observing that the least possible number of differences on diagonal $-k$ is k; hence $D_l[m, m-k]$ is always $\geq k$, and hence $C^k(-k) \leq m$. For $d < 0$, $C^k(d) < m$ implies that $C^{k-1}(d+1) < m$ and hence $C^k(d+1) \leq m$. Hence even if $C^k(d) = \infty$ for some $d \leq 0$, we have $C^k(d') = m$ for some diagonal d' between $-k$ and d. We can therefore determine $m(l)$ by examining all non-∞ values on wave k; hence $m(l) = \max_d\{C^k(d) \mid C^k(d) \leq m\}$, and we wish to find d and h such that $C^h(d) = m(l)$, for this will give us all points $(m(l), m(l)+d)$ for which $D_l[m(l), m(l)+d] = h$.

It suffices to first traverse wave C^k to determine $m(l)$ and thereafter traverse the wave structure in order of diagonals, moving up or down along a diagonal list to find the entry on that diagonal equal to $m(l)$, if any. The algorithm fragment in Figure 11 gives the details. Note that we start by looking for a match with k differences on diagonal $-k$. Thereafter d and h are advanced in unit increments to suggest the pointer-based traversal of the cross-linked structure. The traversal again takes advantage of the fact that adjacent D-values never differ by more than one, implying that if we move from a best point on wave h and diagonal d to the adjacent wave point on the next diagonal, at most one move up or down to wave $h+1$ or wave $h-1$ along that diagonal will reach a point on row $m(l)$ if there is one.

Figure 11 clearly takes $O(k)$ time which is no more expensive than the time to produce the wave structure for the given index l. Thus the overall algorithm takes $O(nk)$ time. Note that a corollary is that there are at most $O(nk)$ k-matches to A. While there are other algorithms for approximate string matching that take only $O(nk)$ time, none delivers or can be trivially extended to deliver the longest prefix of A that can be matched with k differences to some substring of B and to deliver all the k-matching substrings and their associated edit distances. Finding the longest prefix that can be matched to a given string turns out to be crucial in an algorithm for finding approximate repeats, i.e., adjacent substrings whose edit distance is no more than k [LS-93] and is almost certain to find additional applications. Finding all matching substrings is crucial if one has some secondary criteria that is a non-linear function of match length and edit distance. In this case one needs to examine

```
1.   m(l) ← max_d{C^k(d) | C^k(d) ≤ m}
2.   h ← k
3.   for d = −k to k by 1 do
4.    {    if h < k and C^{h+1}(d) ≤ m(l) then
5.              h ← h + 1
6.         else if C^h(d) = ∞ then
7.              h ← h − 1
8.         if C^h(d) = m(l) then
9.              Report A^{m(l)} matches B_l^{l+m(l)+d} with h differences
10.   }
```

FIG. 11. *Reporting matching substrings.*

all matches and not simply the best one ending or beginning at a given index. For example, if $A = aaaacbbbbccccc$ and $B = xxxxxaaaaccccccbbbbccccxxxx\ldots$, then both B_5^{15} and B_{14}^{24} match A (of length 14) with 4 differences but B_5^{24} matches with 5 differences. The last match is conceivably more significant than the two others, (involving 14 identical symbols versus 10), but would not be revealed by previous algorithms. These algorithms can only determine the match(es) with the minimum number of differences ending at a given character of B, or (in a backward solution) the match(es) with the minimum number of differences starting at a given character of B. That is, previous algorithms only detect the difference 4 match B_{14}^{24} to the suffix of B^{24} when run left-to-right over B, and the difference 4 match B_5^{15} to the prefix of B_5 when run right-to-left over B. The difference 5 match B_5^{24} is missed in both directions.

The next application treats this issue in more detail.

4.2. Approximate overlap. Our second example comes from a problem in molecular biology that arises in sequencing DNA. Current methods for determining sequence allow the direct determination of the DNA sequence of a string of length less than 1,000. To determine the sequence of a very long DNA strand, say 50,000 symbols in length, an experimentalist employing the "shotgun" sequencing method randomly extracts fragments of sufficiently small length from the subject strand and determines the sequence of these fragments via a direct experimental method. The resulting *fragment assembly problem* is to determine the subject strand given the collected set of fragments.

The first step in solving the fragment assembly problem is to compare every sequence against every other sequence to see if a suffix of one matches a prefix of another.[4] Such detected overlaps indicate the possibility (but not the certainty) that the two fragments came from overlapping regions of the subject stand. Detecting the overlaps is complicated by the fact that direct sequencing experiments are imperfect and so do not produce the exact sequence. Typically, the error rate runs at about a 1–10% difference between the reported string and the true fragment sequence. Thus, fragments must be compared to determine if there is an *approximate overlap* between them. A fast method is essential for this fundamental subproblem since it must be solved for a quadratic number of fragment pairs.

More precisely, given a threshold k (reflective of the length of the fragments and the error rate of the sequencing method), and two fragments A and B, we say A

[4] [GLS-92] describes an algorithm that finds the exact (0 differences) matches between prefixes and suffixes of a set of strings.

approximately overlaps B within threshold k if either (a) a prefix of A aligns with a suffix of B with not more than k differences, or (b) A aligns with a substring of B with not more than k differences. Matches of type (a) are called *dovetail* matches and those of type (b) are *containment* matches. Unfortunately, whenever A and B approximately overlap within k-difference, there are typically a number of possible ways to do so. One way to compare the significance of different alignments is to choose the overlap which is the least likely to occur by chance, as suggested in [KM-94]. Let $\Pr_\Sigma(m, n, k)$ be the probability, or some approximation thereof, that two strings of respective lengths m and n formed by random Bernoulli trials over a Σ symbol alphabet can be aligned with k-or-less differences. For this discussion, let us assume a precomputed table containing values $Pr_\Sigma[m, n, k]$ between 0 and 1 for the relevant range of m, n, and k (e.g., $0 \leq m, n \leq 2{,}000$, and $0 \leq k \leq 400$ for most DNA sequencing projects). Note that Pr_Σ is typically a nonlinear function of m, n, and k.

Given a threshold k and strings A and B of length m and n, the *approximate overlap problem* can now be stated as finding three indices l, r, and p such that

(a) $r = n$ or $p = m$, and
(b) $ED(A^p, B_l^r) = h \leq k$, and
(c) $Pr_\Sigma[p, r - l, h]$ is minimal.

When $r = n$ the approximate overlap is of the dovetail variety (prefix A^p versus suffix B_l), and when $p = m$ it is of the containment type (A versus substring B_l^r).

As for the approximate match problem, the approximate overlap problem is solved by incrementally computing the solution for A versus B_l for $l = n, n - 1, \ldots, 0$. As before, the initial solution for B_n versus A is easy to compute and $O(k)$ time is spent thereafter incrementally delivering each additional k-thresholded solution as cross-threaded lists of the $k + 1$ waves, $L^0, L^1, \ldots, L^k$. For each suffix B_l, one traverses the wave structure finding all p and r that satisfy conditions (a) and (b) above. We term such a triple, (l, r, p), a *candidate*. As each candidate is discovered, its Pr_Σ "score" is compared against the minimum scoring triple (L, R, P) of probability score S encountered thus far in the algorithm and entered as the new best if its score is lower. Thus, at the end of the algorithm the indices delimiting a minimum scoring approximate overlap and its probability score are delivered. The algorithm is shown in Figure 12.

As for approximate match, the tricky detail is the discovery of the candidates within a given solution. Suppose we have the k-thresholded solution for A versus B_l. Like approximate match, containment candidates correspond to those d and h for which $C^h(d) = m$ for this implies $ED(A, B_l^{l+m+d}) = h$. Dovetail candidates correspond to those furthest points in the structure that reach the extreme column $n - l$ of D_l, the dynamic programming matrix for the problem of comparing A and B_l. That is, we seek d and h for which $C^h(d) = n - l - d$ and is therefore a point on the extreme column of D_l, for this implies $D_l[n - l - d, n - l] = h$ which in turn implies $ED(A^{n-l-d}, B_l) = h$.

Observe that the algorithm given in Figure 12 has exactly the same wave-traversal logic as the one given in Figure 11. The modifications are related to differences between the applications: lines 1 and 8–9 in Figure 11 versus lines 2, 11–14, and 17 in Figure 12. As a consequence, it is clear the algorithm takes $O(nk)$ time.

The algorithm can be further refined to deliver not only the indices of the best approximate overlap in $O(nk)$ time but an alignment achieving it as well. An alignment can of course be produced in all applications but it is particularly appealing here since we output only the "best matching substring" with respect to our scoring function, and we can produce $O(1)$ alignments per substring B_l in the desired time

```
1.  Compute initial solution for A versus B_n.
2.  (L, R, P, S) ← (n, n, 0, 1)    # B_n^n overlaps A^0 with 0 errors with probability 1.
3.  for l = n − 1 downto 0 by 1 do
4.    { Incrementally compute solution for A versus B_l.
5.      h ← k
6.      for d = −k to k by 1 do
7.        { if h < k and C^{h+1}(d) < ∞ then
8.              h ← h + 1
9.          else if C^h(d) = ∞ then
10.             h ← h − 1
11.         if C^h(d) = m or C^h(d) + d = n − l then
12.             s ← Pr_Σ[C^h(d), d + C^h(d), h]
13.             if s < S then
14.                 (L, R, P, S) ← (l, l + d + C^h(d), C^h(d), s)
15.       }
16.   }
17. Best overlap is A^P with B_L^R with score S.
```

FIG. 12. *Approximate overlap algorithm.*

bound. During the traversal of the k-thresholded solution for A versus B_l, record the best candidate encountered in the traversal and if it becomes the best candidate seen thus far in the algorithm, then take $O(k)$ additional time to record the alignment of the overlap modeled by the candidate. In order to take only $O(k)$ additional time, the alignment must be recorded as the ordered sequence of its $O(k)$ *differences*, often called a Δ-encoding of the alignment. Building this encoding simply requires tracing back from the entry $C^h(d)$ corresponding to the candidate. Specifically, from $C^h(d)$ trace back to whichever entry yields the maximum of $C^{h-1}(d-1)$, $C^{h-1}(d)+1$, or $C^{h-1}(d+1)+1$ (by Lemma 2.8), and then trace back from that entry recursively until $C^0(0)$ is reached. If $u = C^{h-1}(d-1)$ gave the maximum, then append "Insert b_{l+u+d}" to the Δ-encoding. If $v = C^{h-1}(d)$ gave the maximum, then append "Substitute $b_{l+v+d+1}$ for a_{v+1}." Finally, if $w = C^{h-1}(d+1)$ gave the maximum, then append "Delete a_{w+1}." Upon completion of the algorithm, the Δ-encoding of the best approximate overlap will have been recorded, and one can use it to produce a display of the alignment in $O(n+m)$ time if desired.

4.3. Cyclic string comparison. Yet another variation of traditional string comparison involves considering *cyclic shifts* of the two strings A and B in question. Let $cycle(a_1a_2\ldots a_m) = a_2\ldots a_ma_1$, and let $cycle^p(A)$ be the result of applying *cycle* exactly p times. The *cyclic string comparison problem* is to determine p and q such that $e = ED(cycle^p(A), cycle^q(B))$ is minimal. It is quite easy to see that if the minimum is obtained for $cycle^p(A)$ and $cycle^q(B)$, then by simply cyclically shifting an alignment one obtains an equally good alignment between A and $cycle^r(B)$ for some r. Thus the problem really reduces to the simpler one of finding q such that $ED(A, cycle^q(B))$ is minimal. This problem was introduced by Mathias Maes [Ma-90] and he gives an $O(mn\log m)$ algorithm for the problem that permits arbitrarily weighted edit costs. Our incremental alignment algorithm leads to a more efficient $O(ne)$ time algorithm for the case of unit cost editing operations.

First we present an $O(n^2)$ algorithm, and later show how to refine it to give an $O(ne)$ algorithm. Consider comparing A and $\overline{B} = B\cdot B$ (B concatenated with itself).

Let the threshold $k = n$ so that for any threshold structure computed $C^k(n-m) \geq m$. Begin by computing the threshold structure for A versus $\overline{B}_n$ $(=B)$ using the greedy algorithm in $O(n^2)$ time. Then incrementally compute the thresholded structure for A versus $\overline{B}_l$, for $l = n-1, n-2 \ldots 0$. We examine each threshold structure to find $ED(A, cycle^l(B))$ in $O(k)$ time. Specifically, it is that h for which $C^h(n-m) = m$ and this is easily found by starting with $C^k(n-m)$ and walking the diagonal list until h is encountered. Note that because k was chosen to be n it follows that h is always on the diagonal list. It takes $O(k)$ time to compute each incremental solution and $O(k)$ additional time to find the edit distance for the given cyclic shift of B. Thus the algorithm takes $O(kn) = O(n^2)$ time to find the minimum edit distance, e, over all possible cyclic shifts.

To bring the complexity down to $O(ne)$ time, consider running the algorithm with a threshold $k < n$. If for a given cyclic shift, $C^k(n-m) < m$, then the edit distance for that cyclic shift of B cannot be determined. On the other hand, if $C^k(n-m) \geq m$ then the edit distance can be computed and k upper bounds the answer e to the overall problem. So consider running the algorithm with $k = 1$, and then with $k = 2$, and $k = 4$, and so on in geometric sequence until the edit distance for at least one cyclic shift is determined in a given trial. Of course, in this last trial, the best edit distance obtained in the trial is the answer, e, to the cyclic string comparison problem. Since k is doubled with each trial, the total time complexity is bounded by the time of the last trial and $k = O(e)$ in that trial. Thus the algorithm takes $O(ne)$ time.

4.4. Text screen updating. Screen oriented programs maintain a representation of an object and present a view of it on the screen. For example, screen editors keep an internal edit buffer and display a block of lines from the buffer. The screen must be updated when the object is changed. In one solution, procedures that modify the object must also update the view or at least specify how the view has changed. A cleaner approach lets an autonomous screen manager module determine how to update the screen by comparing its record of the screen contents with views of the modified object. The interface to the screen manager is then a single routine, *refresh*, that updates the screen with respect to the current object. It is given no information other than the object and screen contents. Unfortunately, the simplicity of the interface requires the screen manager to solve a difficult comparison problem.

The feasibility of this design is demonstrated by the UNIX EMAC editor [Go-81] and the Maryland Window System [Ws-85]. In a two level approach, sequences of lines are compared to decide at the top level which lines to delete, insert, and replace. At the bottom level, the sequences of characters in two lines are compared to appraise and perform line or row replacements. The approach is not guaranteed to update the screen with a minimal set of terminal commands but nonetheless performs well. However, a number of improvements are possible at both levels. At the lower level, Myers and Miller [MM-89] developed algorithms that account for the nonuniformity of terminal command costs and produce optimal update command sequences for the row replacement subproblem. At the top level, a weakness of the earlier approach is the assumption that the screen-sized segment of buffer lines that is to replace the current screen contents is known a priori. More realistically, the screen manager should determine the optimal *window position*, i.e., the screen-sized segment of the buffer that most closely resembles the current screen contents subject to the constraint that the current cursor position be in this segment. A useful approximation to the theoretically optimal choice can be computed economically with our incremental algorithm by finding a window position minimizing the number of screen rows that need to be updated, inserted, or removed.

Suppose B is the current buffer contents, c is the current cursor position, and S is the current screen contents, where B and S are viewed as strings over the infinite alphabet of lines of ASCII text. Suppose B is n lines long and that the screen displays m lines. Formally, the *window positioning problem* is to find a position $p \in [c-m, c-1]$ that minimizes $e = ED(B_p^{p+m}, S)$. Given the cursor position c it is clear that we can restrict our attention to the substring $\overline{B} = B_{c-m}^{c+m-1}$ of B because the window B_p^{p+m} must contain line c. Now observe that this problem is very similar to the one we solved for the cyclic string comparison problem. Namely, we compute solutions for $\overline{B}_l$ for $l = m-1, m-2, \ldots 0$, and for each determine $ED(\overline{B}_l^{l+m}, S)$ which is the value h for which $C^h(0) = m$. Using the same geometric progression of threshold increases, as in the cyclic string comparison problem, gives an $O(me)$ algorithm for the window positioning problem.

Acknowledgment. We would like to thank Esko Ukkonen for bringing the Cyclic String Comparison problem to our attention.

REFERENCES

[BV-93] O. Berkman and U. Vishkin, *Recursive star-tree parallel data-structure,* SIAM J. Comput., 22 (1993), pp. 221–242.

[GP-90] Z. Galil and Q. Park, *An improved algorithm for approximate string matching,* SIAM J. Comput., 19 (1990), pp. 989–999.

[Go-81] J. Gosling, *A redisplay algorithm*, in Proc. ACM SIGPLAN/SIGOA Symposium on Text Manipulation, ACM, New York, 1991, pp. 123–129.

[GLS-92] D. Gusfield, G. M. Landau, and B. Schieber, *An efficient algorithm for the all pairs suffix-prefix problem*, Inform. Process. Lett., 41 (1992), pp. 181–185.

[HD-80] P. A. Hall and G. R. Dowling, *Approximate string matching,* Comput. Surveys, 12 (1980), pp. 381–402.

[HT-84] D. Harel and R. E. Tarjan, *Fast algorithms for finding nearest common ancestors*, SIAM J. Comput., 13 (1984), pp. 338–355.

[Hi-77] D. S. Hirschberg, *Algorithms for the longest common subsequence problem*, J. ACM, 24 (1977), pp. 664–675.

[HS-77] J. W. Hunt and T. G. Szymanski, *An algorithm for differential file comparison*, Comm. ACM, 20 (1977), pp. 350–353.

[KM-94] J. Kececioglu and E. Myers, *Exact and approximate algorithms for the sequence reconstruction problem*, Algorithmica, 13 (1995), pp. 180–210.

[LS-93] G. M. Landau and J. P. Schmidt, *An algorithm for approximate tandem repeats*, in Proc. 4th Symp. Combinatorial Pattern Matching, Lecture Notes in Comput. Sci. 648, Springer-Verlag, New York, 1993, pp. 120–133.

[LV-88] G. M. Landau and U. Vishkin, *Fast string matching with k differences*, J. Comput. System Sci., 37 (1988), pp. 63–78.

[LV-89] G. M. Landau and U. Vishkin, *Fast parallel and serial approximate string matching*, J. Algorithms, 10 (1989), pp. 157–169.

[Ma-90] M. Maes, *On a cyclic string-to-string correction problem*, Inform. Process. Lett., 35 (1990), pp. 73–78.

[Mc-76] E. M. McCreight, *A space-economical suffix tree construction algorithm,* J. ACM, 23 (1976), pp. 262–272.

[My-86a] E. Myers, *An $O(ND)$ difference algorithm and its variations,* Algorithmica, 1 (1986), pp. 251–266.

[My-86b] E. Myers, *Incremental Alignment Algorithms and Their Applications,* Tech. report 86-22, Dept. of Computer Science, University of Arizona, Tucson, AZ, 1986.

[MM-89] E. Myers and W. Miller, *Row replacement algorithms for screen editors*, ACM Trans. Prog. Lang. Systems, 11 (1989), pp. 33–56.

[NKY-82] N. Nakatsu, Y. Kambayashi, and S. Yajima, *A longest common subsequence algorithm suitable for similar text string*, Acta Inform., 18 (1982), pp. 171–179.

[NW-70] S. B. Needleman and C. D. Wunsch, *A general method applicable to the search for similarities in the amino acid sequence of two proteins*, J. Mol. Bio., 48 (1970), pp. 443–453.

[SV-88] B. Schieber and U. Vishkin, *On finding lowest common ancestors: Simplification and parallelization*, SIAM J. Comput., 17 (1988), pp. 1253–1262.

[Se-80] P. H. Sellers, *The theory and computation of evolutionary distances: Pattern recognition,* J. Algorithms, 1 (1980), pp. 359–373.

[SW-81] T. F. Smith and M. S. Waterman, *Identification of common molecular subsequences,* J. Mol. Bio., 147 (1981), pp. 195–197.

[Uk-85a] E. Ukkonen, *Algorithms for approximate string matching,* Inform. Control, 64 (1985), pp. 100–118.

[Uk-85b] E. Ukkonen, *On approximate string matching,* J. Algorithms, 6 (1985), pp. 132–137.

[WF-74] R. A. Wagner and M. J. Fischer, *The string-to-string correction problem,* J. ACM, 21 (1974), pp. 168–173.

[Wn-73] P. Weiner, *Linear pattern matching algorithm,* in Proc. 14th IEEE Symposium on Switching and Automata Theory, IEEE Computer Society Press, Los Alamitos, CA, 1973, pp. 1–11.

[Ws-85] M. Weiser, *CWSH: The windowing shell of the Maryland window system*, Software—Practice and Experience, 15 (1985), pp. 515–519.

[WM-92] S. Wu and U. Manber, *Fast text searching allowing errors*, Comm. ACM, 35 (1992), pp. 83–91.

SIAM J. COMPUT.
Vol. 27, No. 2, pp. 583–604, April 1998
© 1998 Society for Industrial and Applied Mathematics
013

LOCALIZING A ROBOT WITH MINIMUM TRAVEL*

GREGORY DUDEK†, KATHLEEN ROMANIK‡, AND SUE WHITESIDES†

Abstract. We consider the problem of localizing a robot in a known environment modeled by a simple polygon P. We assume that the robot has a map of P but is placed at an unknown location inside P. From its initial location, the robot sees a set of points called the visibility polygon V of its location. In general, sensing at a single point will not suffice to uniquely localize the robot, since the set H of points in P with visibility polygon V may have more than one element. Hence, the robot must move around and use range sensing and a compass to determine its position (i.e., localize itself). We seek a strategy that minimizes the distance the robot travels to determine its exact location.

We show that the problem of localizing a robot with minimum travel is NP-hard. We then give a polynomial time approximation scheme that causes the robot to travel a distance of at most $(k-1)d$, where $k = |H|$, which is no greater than the number of reflex vertices of P, and d is the length of a minimum length tour that would allow the robot to verify its true initial location by sensing. We also show that this bound is the best possible.

Key words. robot, localization, positioning, navigation, sensing, visibility, optimization, NP-hard, competitive strategy

AMS subject classifications. 68Q25, 68T99, 68U05, 68U30

PII. S0097539794279201

1. Introduction.

Numerous tasks for a mobile robot require it to have a map of its environment and knowledge of where it is located in the map. Determining the position of the robot in the environment is known as the *robot localization problem*. To date, mobile robot research that supposes the use of a map generally assumes either that the position of the robot is always known or that it can be estimated using sensor data acquired by displacing the robot only small amounts [4, 24, 30]. However, self-similarities between separate portions of the environment prevent a robot that has been dropped into or activated at some unknown place from uniquely determining its exact location without moving around. This motivates a search for strategies that direct the robot to travel around its environment and to collect additional sensory data [5, 25, 14] to deduce its exact position.

In this paper, we view the general robot localization problem as consisting of two phases: hypothesis generation and hypothesis elimination. The first phase is to determine the set H of *hypothetical locations* that are consistent with the sensing data obtained by the robot at its initial location. The second phase is to determine, in the

*Received by the editors December 23, 1994; accepted for publication (in revised form) March 28, 1996. An earlier version of this paper appeared as McGill University School of Computer Science Technical Report SOCS-94.5 in August 1994. Also, an abridged version of this paper appeared in *Proc. Sixth Annual ACM–SIAM Symposium on Discrete Algorithms*, SIAM, Philadelphia, PA, 1995, pp. 437–446.

http://www.siam.org/journals/sicomp/27-2/27920.html

†Research Centre for Intelligent Machines and School of Computer Science, McGill University, 3480 University Street, Montréal, Québec, Canada H3A 2A7 (dudek@cim.mcgill.ca, sue@cs.mcgill.ca). The research of these authors was supported by NSERC research grants programme.

‡Center for Automation Research, University of Maryland, College Park, MD 20742 (romanik@cfar.umd.edu). This research was done while the author was at McGill University and DIMACS Center for Discrete Mathematics and Theoretical Computer Science. It was supported by IRIS National Network of Centres of Excellence, NSERC, and DIMACS. DIMACS is an NSF Science and Technology Center, funded under contract STC-88-09648 and also receives support from the New Jersey Commission on Science and Technology.

case that H contains two or more locations (see Fig. 2.1), which location is the true initial position of the robot; i.e., to eliminate the incorrect hypotheses.

Ideally, the robot should travel the minimum distance necessary to determine its exact location. This is because the time the robot takes to localize itself is proportional to the distance it must travel (assuming sensing and computation time are negligible in comparison). Also, the most common devices for measuring distance, and hence position, on actual mobile robots are relative measurement tools such as odometers. Therefore, they yield imperfect estimates of orientation, distance, and velocity, and the errors in these estimates accumulate disastrously with successive motions [13]. Our strategy is well suited to handling the accumulation of error problem via simple recalibration, as we will point out later.

A solution to the hypothesis generation phase of robot localization has been given by Guibas, Motwani, and Raghavan in [19]. We describe this further in the next section after making more precise the definitions of the two phases of robot localization. Our paper is concerned with minimizing the distance traveled in the hypothesis elimination phase of robot localization. It begins where [19] left off. Together, the two papers give a solution to the general robot localization problem.

In this paper, we define a natural algorithmic variant of the problem of localizing a robot with minimum travel and show that this variant is NP-hard. We then solve the hypothesis elimination phase with what we call a *greedy localization strategy*. To measure the performance of our strategy, we employ the framework of competitive analysis for on-line algorithms introduced by Sleator and Tarjan [29]. That is, we examine the ratio of the distance traveled by a robot using our strategy to the length d of a minimum length tour that allows the robot to verify its true initial position. The worst case value of this ratio over all maps and all starting points is called the *competitive ratio* of the strategy. If this ratio is no more than k, then the strategy is called *k-competitive*. Since our strategy causes the robot to travel a distance no more than $(k-1)d$, where $k = |H|$ ($|H|$ is no greater than the number of reflex vertices of P), our strategy is $(k-1)$-competitive. We also show that *no* on-line localization strategy has a competitive ratio better than $k-1$, and thus our strategy is optimal.

The rest of this paper is organized as follows. In section 2 we give a formal definition of the robot localization problem, we define some of the terms used in the paper, and we comment on previous work. In section 3 we prove that, given a solution set H to the hypothesis generation phase of the localization problem that contains more than one hypothetical location, the hypothesis elimination phase, which localizes the robot by using minimum travel distance, is NP-hard. In section 4 we define the geometric structures that we use to set up our greedy localization strategy. In section 5 we give our greedy localization strategy and prove the previously mentioned performance guarantee of $k-1$ times optimum. We also give an example of a map polygon for which no on-line localization strategy is better than $(k-1)$-competitive. Section 6 summarizes and comments on open problems.

2. Localization through traveling and probing. In this section, we describe our robot abstraction and give some key definitions.

The most common application domain for mobile robots is indoor "structured" environments. In such environments it is often possible to construct a map of the environment, and it is acceptable to use a polygonal approximation P of the free space [26] as a map. A common sensing method used by mobile robots is range sensing (for example, sonar sensing or laser range sensing).

2.1. Assumptions about the robot. We assume the following throughout this paper.

1. The robot moves in a static two-dimensional, obstacle-free environment for which it has a map. The robot has the ability to make error-free motions between arbitrary locations in the environment.[1] We model the movement of the robot in the environment by a point p moving inside and along the boundary of an n-vertex simple polygon P positioned somewhere in the plane.

2. The robot has a compass and a range sensing device. It is essential that the robot be able to determine its orientation (with the compass); otherwise it can never uniquely determine its exact location in an environment with nontrivial symmetry (such as a square).

3. The robot's sensor can detect the distances to those points on walls for which the robot has an unobstructed straight line of sight, and the robot's observations at a particular location determine a polygon V of points that it can see (see the next subsection for a definition of V). This is analogous to what can be extracted by various real sensors such as laser range finders. The robot also knows its location in V.

2.2. Some definitions and an example. Two points in P are *visible* to each other or *see* each other if the straight line segment joining them does not intersect the exterior of P. The *visibility polygon* $V(p)$ for a point $p \in P$ is the polygon consisting of all points in P that are visible from p. The data received from a range sensing device is modeled as a visibility polygon. The visibility polygon of the initial location of the robot is denoted by V, and the number of its vertices is denoted by m. Since the robot has a compass, we assume that P and V have a common reference direction.

We break the general problem of localizing a robot into two phases as follows.

The robot localization problem.

HYPOTHESIS GENERATION: Given P and V, determine the set H of all points $p_i \in P$ such that the visibility polygon of p_i is congruent under translation to V (denoted by $V(p_i) = V$).

HYPOTHESIS ELIMINATION: Devise a strategy by which the robot can correctly eliminate all but one hypothesis from H, thereby determining its exact initial location. Ideally, the robot should travel a distance as small as possible to achieve this.

As previously mentioned, the hypothesis generation phase has been solved by Guibas, Motwani, and Raghavan. We describe their results in the next subsection. This paper is concerned with the hypothesis elimination phase.

Consider the example illustrated in Fig. 2.1. The robot knows the map polygon P and the visibility polygon V representing what it can "see" in the environment from its present location. Suppose also that it knows that P and V should be oriented as shown. The black dot represents the robot's position in the visibility polygon. By examining P and V, the robot can determine that it is at either point p_1 or point p_2 in P, i.e., $H = \{p_1, p_2\}$. It cannot distinguish between these two locations because $V(p_1) = V(p_2) = V$. However, by traveling out into the "hallway" and taking another probe, the robot can determine its location precisely.

An optimal strategy for the hypothesis elimination phase would direct the robot to follow an optimal verification tour, defined as follows.

[1] In practice, position estimation errors accrue in the execution of such motions; however, the strategy we present here is exceptionally well suited to various methods for limiting these errors using sensor feedback (see section 5.1).

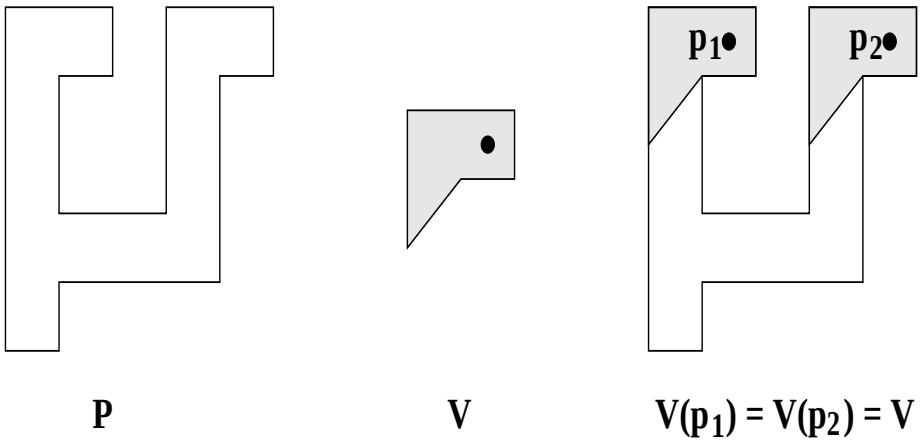


FIG. 2.1. *Given a map polygon P (left) and a visibility polygon V (center), the robot must determine which of the* 2 *possible initial locations p_1 and p_2 (right) is its actual location in P.*

DEFINITION 2.1. *A* verification tour *is a tour along which a robot that knows its initial position* a priori *can travel to verify this information by probing and then return to its starting position. An* optimal verification tour *is a verification tour of minimum length d.*

Since we do not assume *a priori* knowledge of which hypothetical location in H is correct, an optimal verification tour for the hypothesis elimination phase cannot be precomputed. Even if we did have this knowledge, computing an optimal verification tour would be NP-hard. This can be proven using a construction similar to that in section 3 and a reduction from hitting set [16]. For these reasons, we seek an interactive probing strategy to localize the robot. In each step of such a strategy, the robot uses its range sensors to compute the visibility polygon of its present position and from this information decides where to move next to make another probe. To be precise, the type of strategy we seek can be represented by a localizing decision tree, defined as follows.

DEFINITION 2.2. *A* localizing decision tree *is a tree consisting of two kinds of nodes and two kinds of weighted edges. The nodes are either sensing nodes (S-nodes) or reducing nodes (R-nodes), and the node types alternate along any path from the root to a leaf. Thus, tree edges directed down the tree either join an S-node to an R-node (SR-edges) or join an R-node to an S-node (RS-edges).*

1. *Each S-node is associated with a position defined relative to the initial position of the robot. The robot may be instructed to probe the environment from this position.*

2. *Each R-node is associated with a set $H' \subseteq H$ of hypothetical initial locations that have not yet been ruled out. The root is an R-node associated with H, and each leaf is an R-node associated with a singleton hypothesis set.*

3. *Each SR-edge represents the computation that the robot does to rule out hypotheses in light of the information gathered at the S-node end of the edge. An SR-edge does not represent physical travel by the robot and hence has weight* 0.

4. *Each RS-edge has an associated path defined relative to the initial location of the robot. This is the path along which the robot is directed to travel to reach its next sensing point. The weight of an RS-edge is the length of its associated path.*

Since we want to minimize the distance traveled by the robot, we define the weighted height of a localizing decision tree as follows.

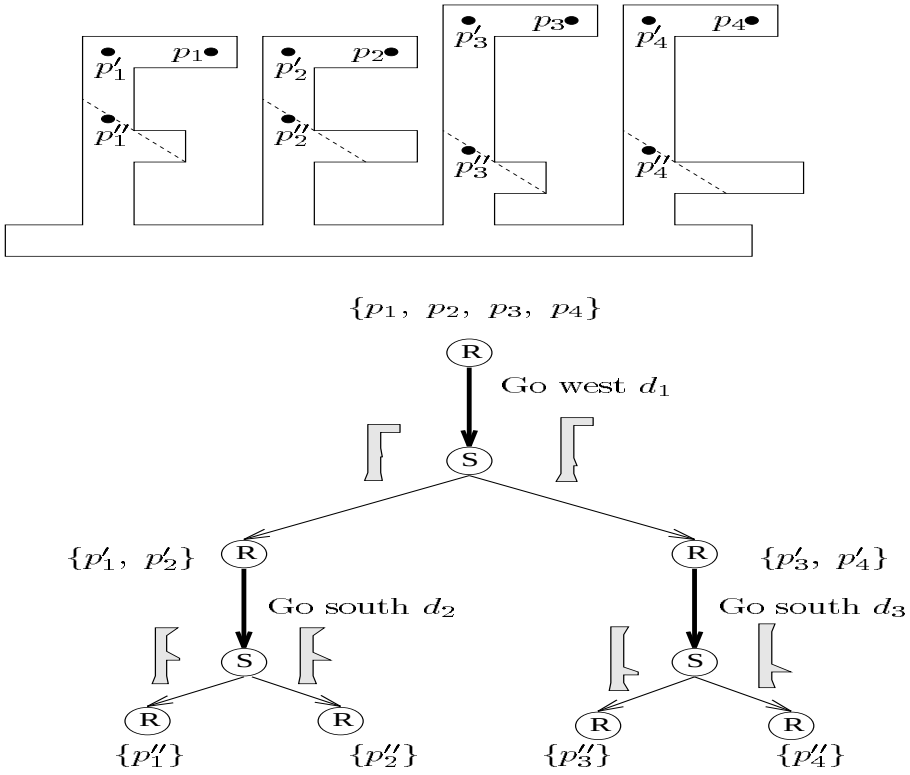


FIG. 2.2. *A map polygon and* 4 *hypothetical locations* $\{p_1, p_2, p_3, p_4\}$ *(above) with a localizing decision tree for determining the true initial position of the robot (below).*

DEFINITION 2.3. *The* weight *of a root-to-leaf path in a localizing decision tree is the sum of the weights on the edges in the path. The* weighted height *of a localizing decision tree is the weight of a maximum-weight root-to-leaf path. An* optimal localizing decision tree *is a localizing decision tree of minimum weighted height.*

In the next section, we show that the problem of finding an optimal localizing decision tree is NP-hard.

We call a localization strategy that can be associated with a localizing decision tree a *localizing decision tree strategy.* As an example of such a strategy, consider the map polygon P shown in Fig. 2.2.

Imagine that, from the visibility polygon sensed by the robot at its initial location, it is determined that the set of hypothetical locations is $H = \{p_1, p_2, p_3, p_4\}$. Hence the root of the localizing decision tree (shown in Fig. 2.2) is associated with H. In the figure, the SR-edges are labeled with the visibility polygons seen by the robot at the S-node endpoints of these edges. Assuming that north points straight up, the strategy given by the tree directs the robot first to travel west a distance d_1, which is the distance between p_i and p'_i, for $1 \leq i \leq 4$, and then to take another probe at its new location. Depending on the outcome of the probe, the robot knows it is located either at one of $\{p'_1, p'_2\}$ or at one of $\{p'_3, p'_4\}$. If it is located at p'_1 or p'_2, then the strategy directs it to travel south a distance d_2, which is the distance between p'_i and p''_i, for $1 \leq i \leq 2$, to a position just past the dotted line segment shown in P. By taking a probe from below this line segment, it will be able to see the vertex at

the end of the segment if it is at location p_1'', and it will not see this vertex if it is at location p_2''. Thus after this probe it will be able to determine its unique location in P. Similarly, if the robot is located at p_3' or p_4', then the strategy directs it to travel south a distance d_3 and take another probe to determine its initial position. The farthest that the robot must travel to determine its location is $d_1 + d_3$, so the weighted height of this decision tree is $d_1 + d_3$.

2.3. Previous work. Previous work on robot localization by Guibas, Motwani, and Raghavan [19] showed how to preprocess a map polygon P so that, given the visibility polygon V that a robot sees, the set of points in P whose visibility polygon is congruent to V, and oriented the same way, can be returned quickly. Their algorithm preprocesses P in $O(n^5 \log n)$ time and $O(n^5)$ space, and it answers queries in $O(m + \log n + k)$ time, where n is the number of vertices of P, m is the number of vertices of V, and k is the size of the output (the number of places in P at which the visibility polygon is V). They also showed how to answer a single localization query in $O(mn)$ time with no preprocessing.

Kleinberg [23] has independently given an interactive strategy for localizing a robot in a known environment. As in our work, he seeks to minimize the ratio of the distance traveled by a robot using his strategy to the length of an optimal verification path (i.e., the competitive ratio). Kleinberg's model differs from ours in several ways. First of all, he models the robot's environment as a geometric tree rather than a simple polygon. A *geometric tree* is a pair (V, E), where V is a finite point set in $\mathbb{R}^d$ and E is a set of line segments whose endpoints all lie in V. The edges E do not intersect except at points of V and do not form cycles. Kleinberg only considers geometric trees with bounded degree Δ. Also, his robot can make no use of vision other than to know the orientation of all edges incident to its current location. Using this model, Kleinberg gives an $O(n^{2/3})$-competitive algorithm for localizing a robot in a geometric tree with bounded degree Δ, where n is the number of branch vertices (vertices of degree greater than two) of the tree.

The competitive ratio of Kleinberg's algorithm appears to be better than the lower bound illustrated by Fig. 5.2 in section 5.3. However, if this map polygon were modeled as a geometric tree it would have degree n, where n is the number of branch vertices, rather than a constant degree, and the distance traveled by a robot using Kleinberg's algorithm can be linear in the degree of the tree. If Kleinberg's algorithm ran on this example, it would only execute step 1, which performs a spiral search, and it would cause the robot to travel a distance almost $4n$ times the length of an optimal verification path. Our algorithm causes the robot to travel a distance less than $2n$ times the length of an optimal verification path on this example. Our algorithm is similar to step 3 of Kleinberg's algorithm, and he gives a lower bound example (Fig. 3 of [23]) illustrating that an algorithm using only steps 1 and 3 of his algorithm is no better than $O(n)$-competitive. Although this example does not directly apply to our model since the robot in our model has the ability to see to the end of the hallway, by adding small jogs in the hallway a similar example can be constructed where our strategy is no better than $O(n)$-competitive. In this example, the number of branch vertices of the geometric tree represented by P would be n and the number of vertices of P would be $O(n)$. However, in this example $|H| = n$, so this does not contradict our results.

Other theoretical work on localizing a robot in a known environment has also been done. Betke and Gurvits [8] gave an algorithm that uses the angles subtended

by landmarks in the robot's environment to localize a robot. Their algorithm runs in time linear in the number of landmarks, and it assumes that a correspondence is given between each landmark seen by the robot and a point in the map of the environment. Avis and Imai [2] also investigated the problem of localizing a robot using angle measurements, but they did not assume any correspondence between the landmarks seen by the robot and points in the environment. Instead they assumed that the environment contains n identical markers, and the robot takes k angle measurements between an unknown subset of these markers. They gave polynomial time algorithms to determine all valid placements of the robot, both in the case where the robot has a compass and where it does not. In addition they showed that, with polynomial-time preprocessing, location queries can be answered in $O(\log n)$ time.

Theoretical work with a similar flavor to ours has also been done on navigating a robot through an unknown environment. In this work a point robot must navigate from a point s to a target t, which is either a point or an infinite wall, where the Euclidean distance from s to t is n. There are obstacles in the scene, which are not known *a priori*, but which the robot learns about only as it encounters them. The goal is to optimize (i.e., minimize) the ratio of the distance traveled by the robot to the length of a shortest obstacle-free path from s to t. As with localization strategies, the worst case ratio over all environments where s and t are distance n apart is called the competitive ratio of the strategy.

Papadimitriou and Yannakakis [28] gave a deterministic strategy for navigating between two points, where all obstacles are unit squares, that achieves a competitive ratio of 1.5, which they show is optimal. For squares of arbitrary size they gave a strategy achieving a ratio of $\sqrt{26}/3$. They also showed, along with Eades, Lin, and Wormald [15], that when t is an infinite wall and the obstacles are oriented rectangles, there is a lower bound of $\Omega(\sqrt{n})$ on the ratio achievable by any deterministic strategy.

Blum, Raghavan, and Schieber [9] gave a deterministic strategy that matched the $\Omega(\sqrt{n})$ lower bound for navigating between two points with oriented, rectangular obstacles. Their strategy combines strategies for navigating from a point to an infinite wall and from a point on the wall of a room to the center of the room, with competitive ratios of $O(\sqrt{n})$ and $O(2^{\sqrt{3 \log n}})$, respectively. The competitive ratio for the problem of navigating from a corner to the center of a room was improved to $O(\ln n)$ by a strategy of Bar-Eli et al. [3], who also showed that this ratio is a lower bound for any deterministic strategy. Berman et al. [7] gave a randomized algorithm for the problem of navigating between two points with oriented, rectangular obstacles with a competitive ratio of $O(n^{4/9} \log n)$.

Several people have studied the problem of navigating from a vertex s to a vertex t inside an unknown simple polygon. They assume that at every point on its path the robot can get the visibility polygon of that point. Klein [21] proved a lower bound of $\sqrt{2}$ on the competitive ratio and gave a strategy achieving a ratio of 5.72 for the class of street polygons. A *street* is a simple polygon such that the clockwise chain L and the counterclockwise chain R from s to t are mutually weakly visible. That is, every point on L is visible to some point on R and vice versa. Kleinberg [22] gave a strategy that improved Klein's ratio to $2\sqrt{2}$, and Datta and Icking [12] gave a strategy with a ratio of 9.06 for a more general class of polygons called *generalized streets*, where every point on the boundary is visible from a point on a horizontal line segment joining L and R. They also showed a lower bound of 9 for this class of polygons.

Previous work in the area of geometric probing has examined the complexity of constructing minimum height decision trees to uniquely identify one of a library of

polygons in the plane using point probes. Such probes examine a single point in the plane to determine if an object is located at that point. If each polygon in the library is given a fixed position, orientation and scale, then it has been shown that both the problem of finding a minimum cardinality probe set (for a noninteractive probing strategy) [6] and the problem of constructing a minimum height decision tree for probing (for an interactive strategy) [1] are NP-complete. Arkin et al. [1] give a greedy strategy that builds a decision tree of height at most $\lceil \log k \rceil$ times that of an optimal decision tree, where k is the number of polygons in the library. The minimum height decision tree used for probing in [1] is different than our localizing decision tree. It is a binary decision tree whose internal nodes represent point probes whose outcome is either positive or negative and whose edges are unweighted. The height of such a decision tree is the number of levels of the tree, and it represents the maximum number of probes necessary to identify any polygon in the library.

3. Hardness of localization. In this section we show that the problem of constructing an optimal localizing decision tree, as defined in the previous section, is NP-hard. To do this, we first formulate the problem as a decision problem.

Robot Localizing Decision Tree (RLDT).

INSTANCE: A simple polygon P and a star-shaped polygon V, both with a common reference direction, the set H of all locations $p_i \in P$ such that $V(p_i) = V$, and a positive integer h.

QUESTION: Does there exist a localizing decision tree of weighted height less than or equal to h that localizes a robot with initial visibility polygon V in the map polygon P?

We show that this problem is NP-hard by giving a reduction from the Abstract Decision Tree problem, proven NP-complete by Hyafil and Rivest in [20]. The Abstract Decision Tree problem is stated as follows.

Abstract Decision Tree (ADT).

INSTANCE: A set $X = \{x_1, \ldots, x_k\}$ of objects, a set $\mathcal{T} = \{T_1, \ldots, T_n\}$ of subsets of X representing binary tests, where test T_j is positive on object x_i if $x_i \in T_j$ and is negative otherwise, and a positive integer $h' \leq n$.

QUESTION: Does there exist an abstract decision tree of height less than or equal to h', where the height of a tree is the maximum number of edges on a path from the root to a leaf, that can be constructed to identify the objects in X?

An abstract decision tree has a binary test at all internal nodes and an object at every leaf. To identify an unknown object, the test at the root is performed on the object, and if it is positive the right branch is taken, otherwise the left branch is taken. This procedure is repeated until a leaf is reached, which identifies the unknown object.

Theorem 3.1. *RLDT is NP-hard.*

Proof. Given an instance of ADT, we create an instance of RLDT as follows. We construct P to be a staircase polygon, with a stairstep for each object $x_i \in X$ (see Fig. 3.1). For each stairstep we construct $n = |\mathcal{T}|$ protrusions, one for each test in $\mathcal{T}$ (see Fig. 3.2). If test T_j is a positive test for object x_i, then protrusion T_j on stairstep x_i has an extra hook on its end (such as T_3, T_4, and T_n in Fig. 3.2). The length of a protrusion is denoted by l and the distance between protrusions T_1 and T_n is denoted by d, where d and l are chosen so that $dh' < l$. The vertical piece between adjacent stairsteps is longer than $(2l + d)h'$, and the width w of each stairstep is much smaller than the other measurements. The polygon P has $O(nk)$ vertices, where $n = |\mathcal{T}|$ and $k = |X|$.

Consider a robot that is initially located at the shaded circle shown in Fig. 3.2 on one of the k stairsteps. The visibility polygon V at this point has $O(n)$ vertices and is

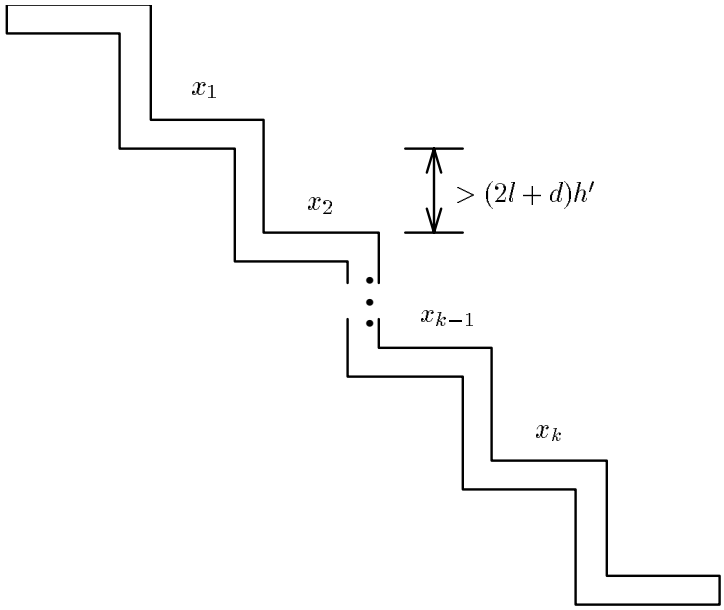


FIG. 3.1. *Construction showing localization is NP-hard.*

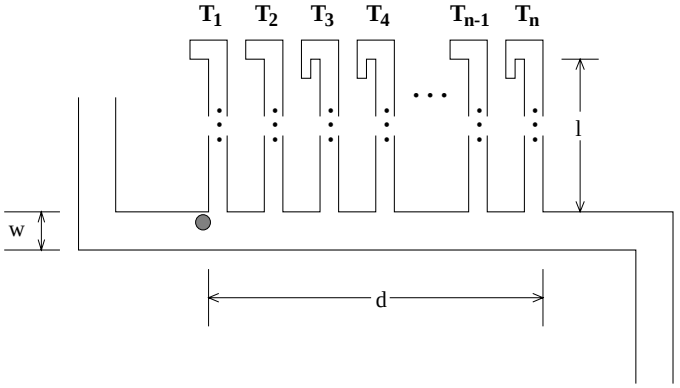


FIG. 3.2. *Close-up of a stairstep x_i in NP-hard construction. Not to scale: $l >> d >> w$.*

the same at an analogous point on any internal stairstep x_i. We output the polygons P and V, which can be constructed in polynomial time, the k locations $p_i \in P$ such that $V(p_i) = V$, and weighted height $h = (2l + d)h'$ as an instance of RLDT.

In order for the robot to localize itself, it must either travel to one of the "ends" of P (either the top or the bottom stairstep) to discover on which stairstep it was located initially, or it must examine a sufficient number of the n protrusions on the stairstep where it is located to distinguish that stairstep from all the others. Since the vertical piece of each stairstep is longer than $h = (2l + d)h'$, only a strategy that directs the robot to remain on the same stairstep can lead to a decision tree of weighted height less than or equal to h.

Any decision tree that localizes the robot by examining protrusions on the stairstep corresponds to an equivalent abstract decision tree to identify the objects of X using

tests in $\mathcal{T}$, and vice versa. Each time the robot travels to the end of protrusion T_j to see if it has an extra hook on its end, it corresponds to performing binary test T_j on an unknown object to observe the outcome. The robot must travel $2l$ to perform this test, and it travels at most d in between tests. Therefore, if the robot can always localize itself by examining no more than h' protrusions, then it has a decision tree of weighted height no more than $h = (2l + d)h'$, which corresponds to an abstract decision tree of height h' for the ADT problem. Since $dh' < l$, in a localizing decision tree of weighted height $\leq h$ the robot cannot examine more than h' protrusions on any root-to-leaf path. □

4. Using a visibility cell decomposition for localization. In this section we discuss the geometric issues involved in building a data structure for our greedy localization strategy.

4.1. Visibility cells and the overlay arrangement. When we consider positions where the robot can move to localize itself, we reduce the infinite number of locations in P to a finite number by first creating a visibility cell decomposition of P [10, 11, 19]. A *visibility cell* (or *visibility region*) C of P is a maximally connected subset of P with the property that any two points in C see the same subset of vertices of P [10, 11]. A *visibility cell decomposition* of P is simply a subdivision of P into visibility cells. This decomposition can be computed in $O(n^3 \log n)$ using techniques in [10, 11]. It is created by introducing $O(nr)$ line segments, called *visibility edges*, into the interior of P, where r is the number of reflex vertices[2] of P. Each line segment starts at a reflex vertex u, ends at the boundary of P, and is collinear with a vertex v that is either visible from u or is adjacent to it. The number of cells in this decomposition, as well as their total complexity, is $O(n^2 r)$ (see [19] for a proof).

Although two points p and q in the same visibility cell C see the same subset of vertices of P, they may not have the same visibility polygon (i.e., it may be that $V(p) \neq V(q)$). This is because some edges of $V(p)$ may not actually lie on the boundary of P (these edges are collinear with p and are produced by visibility lines), so these edges may be different in $V(q)$. Therefore, in order to represent the portion of P visible to a point p in a visibility cell C in such a way that all points in C are equivalent, we need a different structure than the visibility polygon. The structure that we use is the *visibility skeleton* of p.

DEFINITION 4.1. *The* visibility skeleton $V^*(p)$ *of a location $p \in P$ is the skeleton of the visibility polygon $V(p)$. That is, it is the polygon induced by the nonspurious vertices of $V(p)$, where a* spurious *vertex of $V(p)$ is one that lies on an edge of $V(p)$ that is collinear with p, and the other endpoint of this edge is closer to p. The nonspurious vertices of $V(p)$ are connected to form $V^*(p)$ in the same cyclical order that they appear in $V(p)$. The edges of the skeleton are labeled to indicate which ones correspond to real edges from P and which ones are* artificial *edges induced by the spurious vertices. If p is outside P, then $V^*(p)$ is equal to the special symbol $\emptyset$.*

For a complete discussion of visibility skeletons and a proof that $V^*(p) = V^*(q)$ for any two points p and q in the same visibility cell, see [10, 11, 19].

As stated in section 2, the hypothesis generation phase of the robot localization problem generates a set $H = \{p_1, p_2, \ldots p_k\} \subset P$ of *hypothetical locations* at which the robot might be located initially. The number k of such locations is bounded above by r (see [19] for a proof). From this set H, we can select the first location p_1 (or any arbitrary location) to serve as an origin for a local coordinate system. For each

[2] A *reflex vertex* of P is a vertex that subtends an angle greater than 180°.

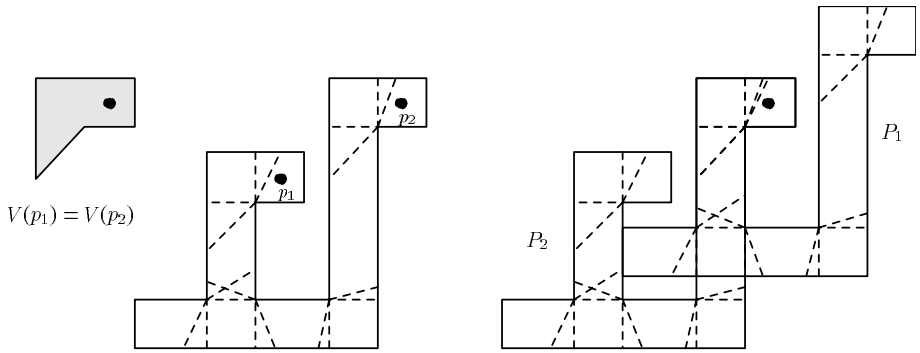


FIG. 4.1. *A visibility polygon, a map polygon and the corresponding overlay arrangement.*

location p_j, $1 \leq j \leq k$, we define the *translation vector* $t_j = p_1 - p_j$ that translates location p_j to location p_1, and we define P_j to be the *translate* of P by vector t_j. We thus have a set $\{P_1, P_2, \ldots, P_k\}$ of translates of P corresponding to the set H of hypothetical locations. The point in each P_j corresponding to the hypothetical location p_j is located at the origin.

In order to determine the hypothetical location corresponding to the true initial location of the robot, we construct an *overlay arrangement* A that combines the k translates P_j that correspond to the hypothetical locations, together with their visibility cell decompositions. More formally, we define A as follows.

DEFINITION 4.2. *The* overlay arrangement *A for the map polygon P corresponding to the set of hypothetical locations H is obtained by taking the union of the edges of each translate P_j as well as the visibility edges in the visibility cell decomposition of P_j.*

See Fig. 4.1 for an example of an overlay arrangement. Since each visibility cell decomposition is created from $O(nr)$ line segments introduced into the interior of P_j, a bound on the total number of cells in the overlay arrangement as well as their total complexity is $O(k^2n^2r^2)$, which may be $O(n^6)$.

4.2. Lower bound on the size of the overlay arrangement. Figure 4.2 shows a map polygon P whose corresponding overlay arrangement for the visibility polygon shown in Fig. 4.3(a) has $\Omega(n^5)$ cells. This polygon has a long horizontal "hallway" with k identical, equally spaced "rooms" on the bottom side of it ($k = 4$ in Fig. 4.2). Each room has width 1 unit, and the distance between rooms is $2k - 1$ units. If the robot is far enough inside one of these rooms so that it cannot see any of the rooms on the top of the hallway, then its visibility polygon is the same no matter which room it is in. The $k - 1$ rooms on the top side of the hallway are identical, have width 1 unit, and are spaced $2k + 1$ units apart. Each top room is between two bottom rooms. The ith top room from the left has its left edge a distance $2i - 1$ to the right of the right edge of the bottom room to its left, and it has its right edge a distance $2(k - i) - 1$ to the left of the left edge of the bottom room to its right (see Fig. 4.2).

Consider the visibility edges starting from the reflex vertices of the bottom rooms that are generated by (i.e., collinear with) the reflex vertices of the top rooms. The ith bottom room from the left will have $2(k - i)$ such visibility edges starting from its right reflex vertex and $2(i - 1)$ starting from its left reflex vertex. Due to the spacing of the top rooms, the visibility edges starting from the reflex vertices of one bottom

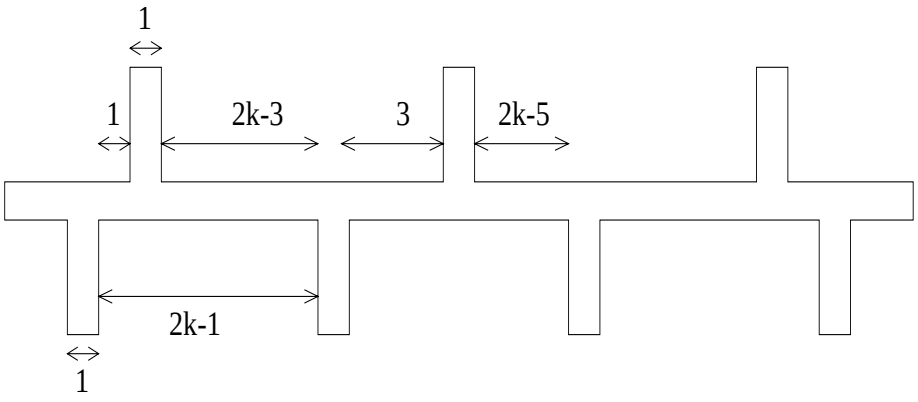


FIG. 4.2. *A map polygon whose overlay arrangement contains* $\Omega(n^5)$ *cells.*

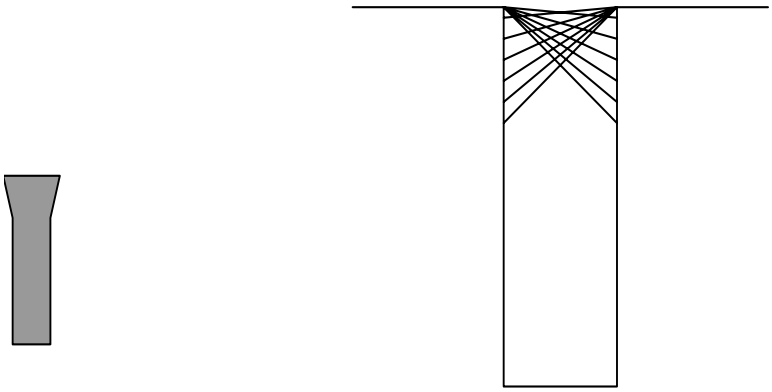


FIG. 4.3. (a) *A visibility polygon.* (b) *Visibility cells in a bottom room.*

room will be at different angles than those in any other bottom room. See the picture in Fig. 4.3(b) for an illustration of the visibility cells inside a bottom room.

When the overlay arrangement A for the visibility polygon shown in Fig. 4.3(a) is constructed, it will consist of k translates, one for each of the bottom rooms of P. Since these rooms are identical and equally spaced, A will have $2k-1$ rooms on its bottom side. Since the visibility edges inside each bottom room are at different angles, these edges will not coincide when bottom rooms from two different translates overlap in A. This means that A will have $\Omega(k)$ bottom rooms with $\Omega(k^2)$ visibility edges starting from the left reflex vertex, and $\Omega(k^2)$ visibility edges starting from the right reflex vertex, resulting in $\Omega(k^4)$ cells inside each of these bottom rooms of A. Therefore, A will have $\Omega(k^5)$ cells in total. Since the number of vertices of P is $8k$, A has $\Omega(n^5)$ cells.

Closing the gap between the upper and lower bounds on the size of the arrangement is an open problem.

4.3. The reference point set Q. Each cell in the overlay arrangement A represents a potential probe position, which can be used to distinguish between different hypothetical locations of the robot. For each cell C of A and for each translate P_j

that contains C, there is an associated visibility skeleton $V_j^*(C)$. If two translates P_i and P_j have different skeletons for cell C, or if C is outside of exactly one of P_i and P_j, then C distinguishes p_i from p_j.

For our localization strategy we choose a set Q of *reference points* in A that will be used to distinguish between different hypothetical locations. For each cell C in A that lies in at least one translate of P, and for each translate P_j that contains C, let $q_{C,j}$ denote the point on the boundary of C that is closest to the origin. Here, the distance $d_j(q_{C,j})$ from the origin to the closest point in C is measured inside P_j. We choose $Q = \{q_{C,j}\}$. In the remainder of this paper we drop the subscripts from $q_{C,j}$ when they are not necessary.

Computing the reference points in Q involves computing Euclidean shortest paths in P_j from the origin to each cell C. To compute these paths we can use existing algorithms in the literature for shortest paths in simple polygons. We first compute for each hypothetical initial location p_j the shortest path tree from the origin to all of the vertices of P_j in linear time using the algorithm given in [18]. This algorithm also gives a data structure for storing the shortest path tree so that the length of the shortest path from the origin to any point $x \in P_j$ can be found in time $O(\log n)$ and the path from the origin to x can be found in time $O(\log n + l)$, where l is the number of segments along this path. We can use this data structure later to extract the shortest path to any cell C in A within any translate P_j.

We use $\pi(p_j, x)$ to denote the shortest path from the origin to x in P_j. To find the shortest path from the origin to a segment $\overline{xy}$ contained in P_j we use the following theorem.

THEOREM 4.3. *If P is a simple polygon, then the Euclidean shortest path $\pi(s, \overline{xy})$ from a point s in P to a line segment $\overline{xy}$ in P is either the shortest path $\pi(s, x)$ from s to x, the shortest path $\pi(s, y)$ from s to y, or a polygonal path with l edges such that the first $l-1$ edges are the first $l-1$ edges on either $\pi(s, x)$ or $\pi(s, y)$, and the last edge is perpendicular to $\overline{xy}$.*

Proof. The theorem follows from standard geometry results. We sketch the proof here. It is shown in [27] that the shortest paths $\pi(s, x)$ and $\pi(s, y)$ are polygonal paths whose interior vertices are vertices of P, and if v is the last common point on these two paths, then $\pi(v, x)$ and $\pi(v, y)$ are both *outward-convex* (i.e., the convex hull of each of these subpaths lies outside the region bounded by $\pi(v, x)$, $\pi(v, y)$ and the segment $\overline{xy}$). As in [27] we call the union $\pi(v, x) \cup \pi(v, y)$ the *funnel* associated with $\overline{xy}$, and we call v the *cusp* of the funnel. See Fig. 4.4 for an example of a simple polygon with edges of this funnel shown as dashed line segments.

The shortest path $\pi(s, \overline{xy})$ has $\pi(s, v)$ as its initial subpath. To complete the shortest path $\pi(s, \overline{xy})$ we must find a shortest path $\pi(v, \overline{xy})$. If v has a perpendicular line of sight to $\overline{xy}$, then this visibility line will be $\pi(v, \overline{xy})$. If v does not have a perpendicular line of sight to $\overline{xy}$, then consider the edge e adjacent to v on the funnel that is the closest to perpendicular. Without loss of generality, assume e is the first edge on $\pi(v, y)$. The path $\pi(v, \overline{xy})$ will follow $\pi(v, y)$ until it reaches y or it reaches a vertex that has a perpendicular line of sight to $\overline{xy}$. □

Using this theorem we can in $O(n)$ time determine the length of the shortest path in P_j from the origin o to $\overline{xy}$ and the closest point on $\overline{xy}$ to o. We first use the data structure in [18] to determine in $O(\log n)$ time the length d_x and the last edge e_x on the shortest path $\pi(o, x)$, and the length d_y and the last edge e_y on the shortest path $\pi(o, y)$. For each of these edges we check its angle with respect to $\overline{xy}$. Note that both of these angles cannot be 90° or greater, or else it would be impossible to form

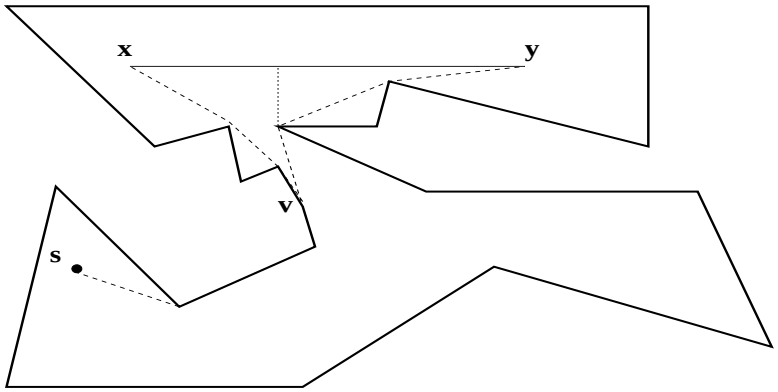


FIG. 4.4. *A simple polygon with shortest paths from s to x, y, and $\overline{xy}$ shown.*

a funnel with $\pi(o,x)$ and $\pi(o,y)$. If the angle between e_x (e_y) and $\overline{xy}$ is at least 90°, then we return d_x (d_y) as the shortest distance to $\overline{xy}$ and x (y) as the closest point on $\overline{xy}$.

If both the angles formed by e_x and e_y with $\overline{xy}$ are less than 90°, then the last edge on the shortest path $\pi(o,\overline{xy})$ will be a perpendicular drawn from one of the vertices on the funnel associated with $\overline{xy}$. To find this edge we again use the data structure in [18] to examine the edges of the funnel in order, starting with e_x. For each edge we calculate the angle formed by its extension with $\overline{xy}$. That is, for each edge (u,w) whose extension intersects $\overline{xy}$ at point z, we calculate the angle $\angle uzy$. As we move around the funnel these angles increase. When the angle becomes greater than 90°, we have found the vertex from which to drop a perpendicular to $\overline{xy}$. It takes $O(n)$ time to find this vertex, and an additional $O(\log n)$ time to calculate the distance to $\overline{xy}$ and the closest point (this is the time it takes to determine the length of the shortest path from o to this vertex).

To compute the reference point $q_{C,j}$, we compute the shortest path distance in P_j from the origin to each edge of C. We then choose the smallest distance as the distance to the cell C. For each cell C we will have up to k reference points $\{q_{C,1},\ldots,q_{C,k}\}$ and their corresponding distances $\{d_1(q_{C,1}),\ldots,d_k(q_{C,k})\}$. We define $d_j(q)=\infty$ for all points q not within P_j.

Partition of H. For each cell C we compute a *partition of H* that represents which hypothetical locations can be distinguished from one another by probing from inside C. If two translates P_i and P_j have the same visibility skeleton for cell C, or if C is outside of both P_i and P_j, then p_i and p_j are in the same subset of the partition of H corresponding to cell C.

Since the visibility polygon and the visibility skeleton for a point $p \in P$ can be computed in $O(n)$ time (see [17]) and we can compare two visibility skeletons with m vertices in $O(m)$ time to see if they are identical, we can compute the partition of H for C in $O(kn+k^2m)$ time, where m is the maximum number of vertices on any of the k visibility skeletons.

Although there may be $O(n^6)$ cells in the overlay arrangement A, yielding up to $O(kn^6)$ reference points, we show in section 5.4 that only $O(k^2)$ reference points are needed for our localization strategy, so we do not need to compute a partition of H for all $O(n^6)$ cells.

5. A greedy strategy for localization. In this section we present a localizing decision tree strategy, called *Minimum Distance Localization Strategy* or *Strategy MDL* for short, for completing the solution of the hypothesis elimination phase of the robot localization problem. Our strategy, which has a greedy flavor, uses the set Q of reference points described in the previous section for choosing probing locations. Strategy MDL has a competitive ratio of $k-1$, where $k = |H|$.

In devising a localizing decision tree strategy, there are two main criteria to consider when deciding where the robot should make the next probe: (1) the distance to the new probe position, and (2) the information to be gained at the new probe position. It is easy to see that a strategy that only considers the second criterion can do arbitrarily worse than an optimal localizing decision tree strategy. Strategy MDL considers (2) only to the extent that it never directs the robot to make a useless probe. Nevertheless, its performance is the best possible. Although it would seem beneficial to weight each possible probe location with the amount of information that could be gained in the worst case by probing at that location, this change will not improve the worst case behavior of Strategy MDL, as the lower bound example given in section 5.3 illustrates.

Even a strategy that considers both the distance and the information criteria when choosing the next probe position can do poorly. For example, if the robot employs an incremental strategy that at each step tells it to travel to the closest probe location that yields some information, then a map polygon can be constructed such that in the worst case the robot will travel distance $2^k d$.

Using Strategy MDL for hypothesis elimination, a strategy for the complete robot localization problem can be obtained as follows. Preprocess the map polygon P using a method similar to that in [19]. This preprocessing yields a data structure that stores for each equivalence class of visibility polygons either the location in P yielding that visibility polygon, if there is only one location, or a localizing decision tree that tells the robot how to travel to determine its true initial location.

5.1. Strategy MDL. In this subsection we present the details of Strategy MDL. Using the results of section 4, it is possible to precompute Strategy MDL's entire decision tree. However, for ease of exposition we will only describe how the strategy directs the robot to behave on a root-to-leaf path in the tree. In practice, it may also sometimes be preferable *not* to precompute the entire tree, but rather to compute the robot's next move on an interactive basis, as the robot carries out the strategy.

Strategy MDL uses the map polygon P, the set H generated in the hypothesis generation phase, and the set Q of reference points defined in section 4.3. Also, for each point $q_{C,j} \in Q$ the strategy uses the distance $d_j(q_{C,j})$ of $q_{C,j}$ from the origin, a path $path_j(q_{C,j})$ within P_j of length $d_j(q_{C,j})$, and the partition of H associated with cell C, as defined in section 4.3.

Next we describe how Strategy MDL directs the robot to behave. Initially, the set of hypothetical locations used by Strategy MDL is the given set H. As the robot carries out the strategy, hypothetical locations are eliminated from H. Thus in our description of Strategy MDL, we abuse notation and use H to denote the shrinking set of *active hypothetical locations*; i.e., those that have not yet been ruled out. Similarly, we use Q to denote the shrinking set of *active reference points*; i.e., those that nontrivially partition the set of active hypothetical locations. We call a path $path_j(q)$ *active* if $p_j \in H$ and $q \in Q$ are both active. We let $d_*(q*)$ denote the minimum of $\{\, d_j(q) \mid q \in Q$ and $p_j \in H$ are active $\}$ and let $path_*(q*)$ denote an active path of length $d_*(q*)$.

Using the initial H and Q, an initial $path_*(q*)$ can be selected. The strategy directs the robot to travel along this path and to make a probe at its endpoint. The robot then uses the information gained at the probe position to update H and Q. The strategy then directs the robot to retrace its path back to the origin and repeat the process until the size of H shrinks to 1.

Note that Strategy MDL is well suited to handling the problem of accumulation of errors caused by successive motions in the estimates of orientation, distance, and velocity made by the robot's sensors. This is because the robot always returns to the origin after making a probe, so it can recalibrate its sensors.

5.2. A performance guarantee for Strategy MDL. The following theorems show that Strategy MDL is correct and has a competitive ratio of $k-1$. First we show that Strategy MDL never directs the robot to pass through a wall. Then we show that Strategy MDL eliminates all hypothetical locations except the valid one while directing the robot along a path no longer than $k-1$ times the length of an optimal verification tour. A corollary of Theorem 5.2 is that the localizing decision tree associated with Strategy MDL has a weighted height that is at most $2(k-1)$ times the weighted height of an optimal localizing decision tree.

THEOREM 5.1. *Strategy MDL never directs the robot to pass through a wall.*

Proof. The proof is by contradiction. Suppose that p_j is the true initial location of the robot and x_j is the point on the boundary of P_j where the robot would first hit a wall. Furthermore, suppose that when the robot attempts to pass through the wall at x_j, the path it has been directed to follow is $path_i(q)$. Let C denote the cell of arrangement A that contains the portion of $path_i(q)$ just before x_j. Since cell C is contained in P_j, it contributes a reference point $q_{C,j}$ to the set Q of reference points.

In order to arrive at a contradiction, it suffices to show that $q_{C,j}$ is active at the time Strategy MDL chooses $path_i(q)$ for the robot to follow. This is because $d_j(q_{C,j}) \le d_j(x_j)$ by definition of $q_{C,j}$, $d_j(x_j) \le d_i(x_j)$ since the portion of $path_i(q)$ from the origin to x_j is contained within P_j, and $d_i(x_j) < d_i(q)$ because x_j is an intermediate point on $path_i(q)$. Thus $d_j(q_{C,j}) < d_i(q)$, so Strategy MDL would choose $path_j(q_{C,j})$ rather than $path_i(q)$ if $q_{C,j}$ is active.

Point $q_{C,j}$ is active when $path_i(q)$ is selected because cell C distinguishes between the two active hypothetical locations p_i and p_j. This is because the skeleton $V_j^*(C)$ associated with C relative to P_j has a real edge through the point x_j, whereas the skeleton $V_i^*(C)$ associated with C relative to P_i does not have a real edge through x_j. □

THEOREM 5.2. *Strategy MDL localizes the robot by directing it along a path whose length is at most $(k-1)d$, where $k = |H|$ and d is the length of an optimal verification tour for the robot's initial position.*

Proof. Let p_t denote the true initial location of the robot. First we show by contradiction that Strategy MDL eliminates all hypothetical initial locations in H except p_t. Suppose Q becomes empty before the size of H shrinks to one, and let p_i be an active hypothetical location different from p_t at the time Q becomes empty. Translates P_i and P_t are not identical, so there is some point x_t on the boundary of P_t that does not belong to the boundary of P_i. Let C be the cell of arrangement A contained in P_t and containing x_t. C distinguishes between p_i and p_t, so $q_{C,t}$ is still in the active set Q — a contradiction.

Next we establish an upper bound on the length of the path determined by Strategy MDL. Because the strategy always directs the robot to a probing site that eliminates one or more elements from H, the robot makes at most $k-1$ trips from its

initial location to a sensing point and back. To show that each round trip has length at most d, we consider how a robot traveling along an optimal verification tour L would rule out an arbitrary incorrect hypothetical location p_i. Then we consider how Strategy MDL would rule out p_i.

Consider a robot traveling along tour L that eliminates each invalid hypothetical location at the first point x on L where the visibility skeleton of x relative to the invalid hypothetical location differs from the visibility skeleton of x relative to P_t.

Let x be the first point on L where the robot can eliminate p_i. The point x must lie on the boundary of some cell C in the arrangement A that distinguishes p_i from p_t. Cell C generates a reference point $q_{C,t} \in Q$, and $d_t(q_{C,t}) \leq d_t(x)$. Since p_t is the true initial location of the robot, the distance $d_t(x)$ is no more than the distance along L of x from the origin, as well as the distance along L from x back to the origin. Thus $d_t(q_{C,t})$ is no more than half the length of L.

At the moment Strategy MDL directs the robot to move from the origin to the probing site where it eliminates p_i, both p_i and p_t are active, so point $q_{C,t}$ is active since it distinguishes between them. At this time Strategy MDL directs the robot to travel along $path_*(q*)$. By definition, the length $d_*(q*)$ of this path is the minimum over all $d_j(q)$ for active $p_j \in H$ and $q \in Q$. In particular, since point $q_{C,t}$ is still active, $d_*(q*) \leq d_t(q_{C,t})$, which is no more than half the length of L. Therefore, Strategy MDL directs the robot to travel along a loop from the origin to some probing position where the robot eliminates p_i and back, and the length of this loop is at most d. □

Using the definition of competitive ratio given in section 1, Theorem 5.2 can be stated as "Strategy MDL is $(k-1)$-competitive, where $k = |H|$." Note that if a verifying path is not required to return to its starting point, the bound for Theorem 5.2 becomes $2(k-1)d$. Note also that even if the robot were continuously sensing rather than just taking a probe at the end of each path $path_*(q*)$, a better bound could not be achieved. This is because the robot always goes to the closest point that yields useful information, so no point on $path_*(q*)$ before $q*$ will allow it to eliminate any hypothetical locations.

COROLLARY 5.3. *The weighted height of the localizing decision tree constructed by Strategy MDL is at most* $2(k-1)$ *times the weighted height of an optimal localizing decision tree for the same problem.*

Proof. Consider the decision tree of Strategy MDL. Let p_h denote the initial location associated with the leaf that defines the weighted height of the tree. The weighted height of the tree is thus the distance Strategy MDL will direct the robot to travel to determine that p_h is the correct initial location, and by Theorem 5.2 this distance is at most $k-1$ times the minimum verification tour length for p_h. But the minimum verification tour length for p_h is at most twice the weight of a path from the root to p_h in an optimal localizing decision tree, which is at most the weighted height of the tree. The result follows from these inequalities. □

If the robot is required to return to its initial position, the bound on the weighted height of the localizing decision tree constructed by Strategy MDL drops to $k-1$.

It should be clear from the discussions in sections 4 and 5 that Strategy MDL can be computed and executed in polynomial time. In this paper, we do not comment further on computation time as there are many ways to implement Strategy MDL. Also, if travel times are large compared to computation times, the importance of our results is that they obtain good path lengths.

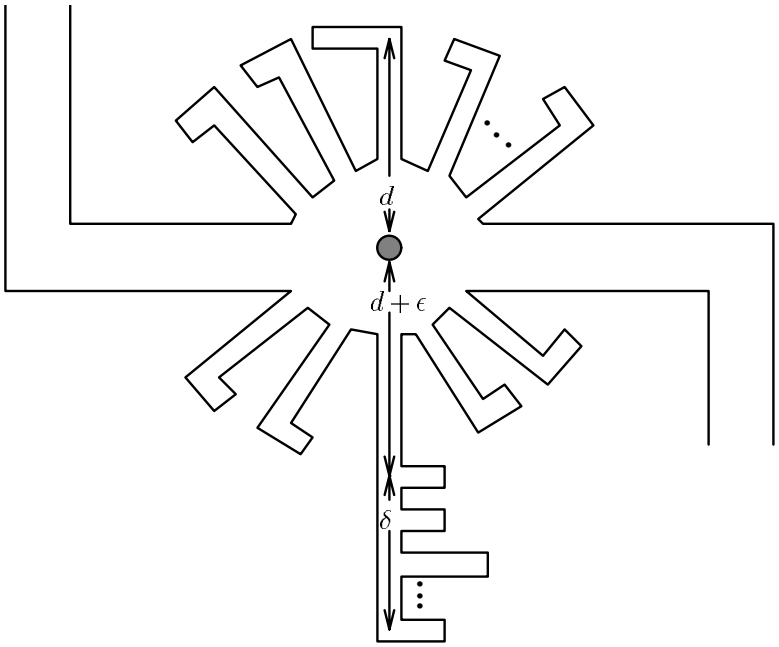


FIG. 5.1. *Part of the map polygon that gives lower bound. Not to scale:* $d >> \epsilon, \delta$.

5.3. Lower bounds. In Corollary 5.3 we proved that the weighted height of the localizing decision tree built by Strategy MDL is no greater than $2(k-1)d$, where $k = |H|$ and d is the weighted height of an optimal localizing decision tree. This bound is also a lower bound for Strategy MDL, as illustrated in Fig. 5.1. Consider a map polygon that is a staircase polygon with $k+2$ stairs, such as the one in Fig. 3.1, where each stairstep except the first and last one is similar to the one shown in Fig. 5.1. Each such stairstep has k protrusions placed in a circle, with the end of each protrusion a distance d from the center of the circle. In each stairstep a different protrusion has its end extended, which uniquely identifies the stairstep. Each stairstep also has a longer protrusion, with k smaller protrusions sticking out of it. One of these smaller protrusions is extended to uniquely identify the stairstep. The first small protrusion is a distance $d+\epsilon$ from the center of the circle, and the last one is a distance $d+\epsilon+\delta$ from the center of the circle.

For this map polygon, if the robot is initially placed in the center of the circle on one stairstep, Strategy MDL will direct it to travel up the k protrusions of length d until it finds one that has a longer piece at the end, or until it has examined all but one of these protrusions. In the worst case the robot will travel a distance $2(k-1)d$. An optimal strategy would direct the robot to travel down the protrusion of length $d+\epsilon+\delta$ and examine all the small protrusions coming out of it until it found one that was longer. In the worst case the robot would travel a distance $d+\epsilon+\delta$. Since ϵ and δ can be made arbitrarily small, in the worst case Strategy MDL travels $\Omega(k)$ times as far as the optimal strategy. Even if we used a strategy that weighted each potential probe location with the amount of information that could be gained from that location in the worst case, we would still build the same decision tree because any probe location in the stairstep yields at most one piece of information in the worst case.

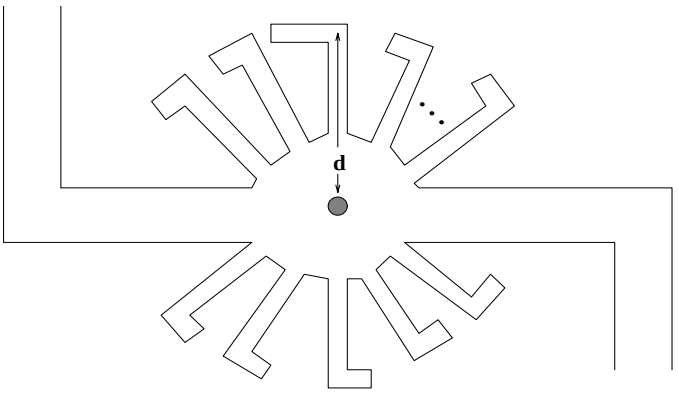


FIG. 5.2. *Part of the map polygon that shows Strategy MDL is best possible.*

Although there are map polygons for which Strategy MDL builds a localizing decision tree whose weighted height is $\Omega(k)$ times the weighted height of an optimal localizing decision tree, there are other map polygons for which any localizing decision tree strategy builds a tree with weighted height at least $k-1$ times the length of an optimal verification tour. Consider a map polygon that is a staircase polygon with $k+2$ stairs, such as the one in Fig. 3.1, where each stairstep except the first and last one is similar to the one shown in Fig. 5.2. Each such stairstep has k protrusions placed in a circle, with the end of each protrusion a distance d from the center of the circle, and has one protrusion extended at the end to uniquely identify the stairstep. The vertical piece between adjacent stairsteps is longer than $2(k-1)d$.

As with the map polygon shown in Fig. 5.1, Strategy MDL will direct the robot to explore the k protrusions of length d, and in the worst case the robot will travel a distance $2(k-1)d$. Consider any other localizing decision tree strategy. If it directs the robot to travel to any stairstep besides the one where it starts, then the localizing decision tree that it builds will have weighted height greater than $2(k-1)d$. The only way to localize the robot while remaining on the initial stairstep is to direct it to examine the protrusions, and in the worst case the robot must travel a distance $2(k-1)d$ before it has localized itself (assuming that it must return to the origin at the end).

Since no localizing decision tree strategy can build a tree with weighted height less than $k-1$ times the length of an optimal verification tour for all map polygons, Strategy MDL is the best possible strategy.

5.4. Creating a reduced set of reference points. The set Q of reference points has size upper bounded by k times the number of cells in the arrangement A, which may be very large as shown in section 4.2. In this subsection, we show that when Strategy MDL is run with only a small subset $Q' \subseteq Q$ of the original reference points, the $(k-1)d$ performance guarantee of section 5.2 still holds. The size of Q' will be no more than $k(k-1)$.

Set Q' is defined as the union of subsets $Q_i \subseteq Q$, where there is one Q_i for each $p_i \in H$ and $|Q_i| \leq k-1$. Ignoring implementation issues, we define Q_i as follows. Initially Q_i is empty, and the subset of Q consisting of reference points $q_{C,i}$ generated for translate P_i is processed in order of increasing $d_i(q_{C,i})$. For each successive point $q_{C,i}$, the partition of H induced by $Q_i \cup \{q_{C,i}\}$ is compared to that induced by Q_i

alone. If the subset of H containing location p_i is further subdivided by the additional reference point $q_{C,i}$, then $q_{C,i}$ is added to Q_i. Conceptually, the reference point $q_{C,i}$ is added if it distinguishes another hypothetical initial location from p_i. This process continues until p_i is contained in a singleton in the partition of H induced by Q_i. Since there are only $k-1$ initial locations to be distinguished from p_i, Q_i will contain at most $k-1$ points.

We denote by *Strategy MDLR*, which stands for *Minimum Distance Localization with Reduced reference point set*, the strategy obtained by replacing set Q with Q' in Strategy MDL.

THEOREM 5.4. *Strategy MDLR, which uses a set of at most $k(k-1)$ reference points, localizes the robot by directing it along a path whose length is at most $(k-1)d$, where $k = |H|$ and d is the length of an optimal verification tour for the robot's initial position.*

Proof. Both the proof that Strategy MDLR directs the robot along a path that determines its initial location and the proof of the $(k-1)d$ bound are essentially the same as the proofs of the corresponding results in Theorems 5.1 and 5.2 of section 5.2. The only additional observation needed is that if a reference point $q_{C,i}$ is used in one of the previous proofs to distinguish between two hypothetical initial locations, and if $q_{C,i}$ does not belong to set Q', then Q' contains some reference point $q_{C',j}$ that distinguishes the same pair of locations and that satisfies $d_j(q_{C',j}) \leq d_i(q_{C,i})$. Hence, set Q' always contains an adequate substitute for any reference point of Q required by the proofs of Theorems 5.1 and 5.2. □

6. Conclusions and future research. We have shown that the problem of localizing a robot in a known environment by traveling a minimum distance is NP-hard, and we have given an approximation strategy that achieves a competitive ratio of $k-1$, where k is the number of possible initial locations of the robot. We have also shown that this bound is the best possible.

The work in this paper is one part of a strategy for localizing a robot. The complete strategy will preprocess the map polygon and store the decision trees for ambiguous initial positions so that the robot only needs to follow a predetermined path to localize itself.

There are many variations to this problem which can be considered. If the robot must localize itself in an environment with obstacles, then the map of the environment can be represented as a simple polygon with holes. If these obstacles are moving, then the problem becomes more difficult.

In this paper we assigned a cost of zero for the robot to take a probe and analyze it. In a more general setting we would look for an optimal decision tree, where the edges of a decision tree associated with the outcome of a probe would be weighted with the cost to analyze that probe. A pragmatic variation of the problem would weight reference locations so that those that produce more reliable percepts would be selected first.

REFERENCES

[1] E. M. ARKIN, H. MEIJER, J. S. MITCHELL, D. RAPPAPORT, AND S. S. SKIENA, *Decision trees for geometric models*, in Proc. 9th Annual ACM Symposium on Computational Geometry, San Diego, CA, May 19-21, 1993, ACM, New York, pp. 369–378.

[2] D. AVIS AND H. IMAI, *Locating a robot with angle measurements*, J. Symbolic Comput., 10 (1990), pp. 311–326.

[3] E. Bar-Eli, P. Berman, A. Fiat, and P. Yan, *On-line navigation in a room*, in Proc. 3rd Annual ACM–SIAM Symposium on Discrete Algorithms, Orlando, FL, January 27–29, 1992, SIAM, Philadelphia, pp. 237–249.

[4] R. Basri and E. Rivlin, *Homing using combinations of model views*, in Proc. 13th Internat. Joint Conference on Artificial Intelligence (IJCAI-93), Chambery, France, August 1993, Morgan Kaufmann Publishers, San Francisco, CA, pp. 1586–1591.

[5] K. Basye and T. Dean, *Map learning with indistinguishable locations*, in Uncertainty in Artificial Intelligence 5, M. Henrion, L. N. Kanal, and J. F. Lemmer, eds., Elsevier Science Publishers, New York, 1990, pp. 331–340.

[6] P. Belleville and T. C. Shermer, *Probing polygons minimally is hard*, Comput. Geom., 2 (1993), pp. 255–265.

[7] P. Berman, A. Blum, A. Fiat, H. Karloff, A. Rosen, and M. Saks, *Randomized robot navigation algorithms*, in Proc. 7th Annual ACM–SIAM Symposium on Discrete Algorithms, Atlanta, GA, January 28–30, 1996, SIAM, Philadelphia, pp. 75–84.

[8] M. Betke and L. Gurvits, *Mobile robot localization using landmarks*, in Proc. IEEE/RSJ/GI Internat. Conference on Intelligent Robots and Systems, Munich, Germany, September 1994, IEEE Computer Society Press, Los Alamitos, CA, pp. 135–142. To appear in IEEE Trans. on Robotics and Automation.

[9] A. Blum, P. Raghavan, and B. Schieber, *Navigating in unfamiliar geometric terrain*, in Proc. 23rd Annual ACM Symposium on Theory of Computing, ACM, New York, 1991, pp. 494–504; SIAM J. Comput., 26 (1997), pp. 110–137.

[10] P. K. Bose, *Visibility in Simple Polygons*, Master's thesis, University of Waterloo, Waterloo, Ontario, Canada, December, 1991.

[11] P. K. Bose, A. Lubiw, and J. I. Munro, *Efficient visibility queries in simple polygons*, in Proc. 4th Canadian Conference on Computational Geometry, C. A. Wang, ed., St. John's, Newfoundland, Canada, August 10-14, 1992, Memorial University of Newfoundland, pp. 23–28.

[12] A. Datta and C. Icking, *Competitive searching in a generalized street*, in Proc. 10th Annual ACM Symposium on Computational Geometry, Stony Brook, NY, June 6–8, 1994, ACM Press, New York, pp. 175–182.

[13] E. Davis, *Representing and Acquiring Geographic Knowledge*, Pitman and Morgan Kaufmann Publishers, Inc., London and Los Altos, CA, 1986.

[14] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes, *Map validation and self-location in a graph-like world*, in Proc. 13th International Joint Conference on Artificial Intelligence (IJCAI-93), Chambery, France, August, 1993, Morgan Kaufmann Publishers, San Francisco, CA, pp. 1648–1653.

[15] P. Eades, X. Lin, and N. Wormald, *Performance guarantees for motion planning with temporal uncertainty*, Austral. Comput. J., 25 (1993), pp. 21–28.

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

[17] H. A. E. Gindy and D. Avis, *A linear algorithm for computing the visibility polygon from a point*, J. Algorithms, 2 (1981), pp. 186–197.

[18] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan, *Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons*, Algorithmica, 2 (1987), pp. 209–233.

[19] L. J. Guibas, R. Motwani, and P. Raghavan, *The robot localization problem*, SIAM J. Comput., 26 (1997), pp. 1120–1138.

[20] L. Hyafil and R. L. Rivest, *Constructing optimal binary decision trees is NP-complete*, Inform. Process. Lett., 5 (1976), pp. 15–17.

[21] R. Klein, *Walking an unknown street with bounded detour*, Comput. Geom., 1 (1992), pp. 325–351.

[22] J. Kleinberg, *On-line search in a simple polygon*, in Proc. Fifth Annual ACM–SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1994, pp. 8–15.

[23] J. Kleinberg, *The localization problem for mobile robots*, in Proc. 35th Annual IEEE Symposium on Foundations of Computer Science, Santa Fe, NM, November 20–22, 1994, IEEE Computer Society Press, Los Alamitos, CA, pp. 521–533.

[24] A. Kosaka, M. Meng, and A. C. Kak, *Vision-guided mobile robot navigation using retroactive updating of position uncertainty*, in Proc. IEEE Internat. Conference on Robotics and Automation, Volume 2, Atlanta, GA, May, 1993, IEEE Computer Society Press, Los Alamitos, CA, pp. 1–7.

[25] B. J. Kuipers and Y. T. Byun, *A qualitative approach to robot exploration and map-learning*, in Proc. IEEE Workshop on Spatial Reasoning and Multi-Sensor Fusion, Los Altos, CA, 1987, IEEE Computer Society Press, Los Alamitos, CA, pp. 390–404.

[26] J.-C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, Norwell, MA, 1991.
[27] D. Lee and F. P. Preparata, *Euclidean shortest paths in the presence of rectilinear barriers*, Networks, 14 (1984), pp. 393–410.
[28] C. Papadimitriou and M. Yannakakis, *Shortest paths without a map*, Theoret. Comput. Sci., 84 (1991), pp. 127–150.
[29] D. D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Comm. ACM, 28 (1985), pp. 202–208.
[30] R. Talluri and J. K. Aggarwal, *Position estimation for an autonomous mobile robot in an outdoor environment*, IEEE Trans. on Robotics and Automation, 8 (1992), pp. 573–584.

SIAM J. COMPUT.
Vol. 27, No. 3, pp. 605–613, June 1998
© 1998 Society for Industrial and Applied Mathematics

LISTING ALL MINIMAL SEPARATORS OF A GRAPH*

T. KLOKS† AND D. KRATSCH‡

Abstract. An efficient algorithm listing all minimal vertex separators of an undirected graph is given. The algorithm needs polynomial time per separator that is found.

Key words. graph, algorithm, listing algorithm, minimal separator, good pair, minimal pair

AMS subject classifications. 68R10, 05C85

PII. S009753979427087X

1. Introduction. Given a graph, one is often interested in listing certain subsets of vertices, or their cardinality, which possess a certain property. For example, the CLIQUE NUMBER of a graph G is the maximum cardinality of a subset S such that $G[S]$ is complete. Similar questions are the INDEPENDENCE NUMBER, the DOMINATION NUMBER, or the CHROMATIC NUMBER. For many of these problems, it would be convenient if one could use a decomposition of the graph by means of certain *separators*.

This is perhaps best illustrated by the recent results for classes of graphs of bounded treewidth. For these classes, linear time algorithms exist for many NP-complete problems *exactly* because a decomposition can be made using *separators of bounded size* [1, 2, 3, 4, 12]. A decomposition of this type can be found in linear time [5, 12]; however, the huge constants involved in these algorithms do not make them of much practical use.

A closely related but somewhat different approach was surveyed in [19]. In that paper (see also [8]) it is shown that for many classes of graphs (for example chordal graphs, clique separable graphs, and edge intersection graphs of paths in a tree (EPT-graphs)) a decomposition by *clique separators* is possible, and it is illustrated that such a decomposition can also be used to solve efficiently many NP-complete problems like MINIMUM FILL-IN, MAXIMUM CLIQUE, GRAPH COLORING, and MAXIMUM INDEPENDENT SET. In [20] an algorithm is given for finding clique separators efficiently (the algorithm uses $O(nm)$ time to find one clique separator). Recent results have shown how the above-mentioned results can be generalized in the sense that at least the NP-complete problems TREEWIDTH and MINIMUM FILL-IN can be solved by polynomial time algorithms for many more graph classes, i.e., certain graph classes for which the number of minimal separators is polynomial bounded [6, 7, 12, 13, 14, 15, 17].

In [11] an algorithm is given which finds all of what the authors call *minimum size separators*. By this they mean that, given a graph which is k-connected, the algorithm finds all separators with k vertices. Moreover, they show in this paper that the number of these separators is bounded by $O(2^k \frac{n^2}{k})$. Their algorithm, which lists all minimum size separators, runs in $O(2^k n^3)$ time. We call a subset of vertices S a

*Received by the editors July 11, 1994; accepted for publication (in revised form) February 1, 1996. A preliminary version of this paper appeared in *Proceedings of the* 11*th Annual Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science 775, Springer-Verlag, Berlin, 1995, pp. 759–768.

http://www.siam.org/journals/sicomp/27-3/27087.html

†Faculty of Applied Mathematics, University of Twente, P.O. Box 217, AE Enschede, The Netherlands (ton@win.tue.nl).

‡Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität, 07740 Jena, Germany (dieter.kratsch@mathematik.uni-jena.dbp.de).

minimal separator if there are nonadjacent vertices x and y such that the removal of S separates x and y into disjoint connected components in such a way that no proper subset of S also does this (see Definition 1). A closely related concept which we call *inclusion minimal separators* lies more or less between the minimum size separators and the minimal separators, i.e., all minimum size separators are inclusion minimal and all inclusion minimal separators are minimal separators.

The following example shows that the minimum size separators and the inclusion minimal separators are only of limited use. Consider any graph G. Take a new vertex x and make this adjacent to all vertices of G. Take another new vertex y and make this adjacent to x. Call this new graph H. The only inclusion minimal separator which is also the only minimum size separator of H is $\{x\}$. However, if S is some minimal separator of G, then $S \cup \{x\}$ is a minimal separator in H. Hence H has at least as many minimal separators as G.

In [6, 12, 13, 14, 15] it is shown that many important classes of graphs have a polynomial number of minimal vertex separators. These graph classes include permutation graphs, circular permutation graphs, trapezoid graphs, circle graphs, circular arc graphs, distance hereditary graphs, chordal bipartite graphs, cocomparability graphs of bounded dimension, and weakly triangulated graphs. For some of these graph classes there are efficient algorithms listing all minimal vertex separators, often using so-called "scanlines" (see, e.g., [6, 12, 17]). In this paper we present an algorithm for listing all minimal vertex separators of any given graph. Notice that, in general, the number of minimal separators can be exponential, as the following example shows. Consider the graph consisting of two nonadjacent vertices s and t and a set of $\frac{n-2}{2}$ (internally) vertex disjoint paths of length 3 from s to t. The number of minimal s,t-separators in this graph is $2^{(n-2)/2}$.

Our listing algorithm has been applied as an important subroutine in $O(n^5R + n^3R^3)$ algorithms computing the treewidth and minimum fill-in of a given asteroidal triple-free graph with n vertices and R minimal separators [16, 17]. (Notice that asteroidal triple-free graphs are a relatively large class of graphs containing cocomparability graphs and permutation graphs.) Furthermore, it has been suggested in [18] to use a so-called "separator graph" for obtaining polynomial time treewidth and minimum fill-in algorithms. Thereby the vertex set of the separator graph is the set of all minimal separators of the given graph. Typically, applications require our listing algorithm.

For listing other types of combinatorial structures we refer to [9].

2. Preliminaries. If $G = (V, E)$ is a graph and $W \subseteq V$ is a subset of vertices, then we use $G[W]$ as a notation for the subgraph of G *induced* by the vertices of W. For a vertex $x \in V$ we use $N(x)$ to denote the neighborhood of x.

The following definition can be found, for example, in [10].

DEFINITION 1. *Given a graph $G = (V, E)$ and two nonadjacent vertices a and b, a subset $S \subset V$ is an* a,b-separator *if the removal of S separates a and b in distinct connected components. If no proper subset of S is an a,b-separator, then S is a* minimal a,b-separator. *A* minimal separator *is a set of vertices S for which there exist nonadjacent vertices a and b such that S is a minimal a,b-separator.*

The following lemma appears, for example, as an exercise in [10]. It provides an easy test of whether or not a given set S of vertices is a minimal separator.

LEMMA 2. *Let S be a separator of the graph $G = (V, E)$. Then S is a minimal separator if and only if there are two different connected components of $G[V - S]$ such that every vertex of S has a neighbor in both of these components.*

Proof. Let S be a minimal a,b-separator and let C_a and C_b be the connected components containing a and b, respectively. Let $x \in S$. Since S is a *minimal* a,b-separator, there is a path between a and b passing through x but using no other vertex in S. Hence x must have a neighbor in C_a and C_b.

Now let S be a separator and let C_a and C_b be two connected components such that every vertex of S has a neighbor in C_a and C_b. Let $a \in C_a$ and $b \in C_b$. Then, clearly, S is a minimal a,b-separator, for if $x \in S$, then there is a path between a and b which uses no vertices of $S \setminus \{x\}$. □

Notice that this also proves the following. Let S be a minimal separator and let C_1 and C_2 be two connected components of $G[V-S]$ such that every vertex of S has a neighbor in both C_1 and C_2. If a is a vertex of C_1 and b is a vertex of C_2, then S is a minimal a,b-separator.

It may be a bit surprising at first to see that it is very well possible for one minimal separator to be contained in another. An example of this can be found in [10]. However, for minimal a,b-separators things are different since, by definition, one minimal a,b-separator cannot be properly contained in another.

We now show that at least some of the minimal separators are easy to find.

DEFINITION 3. *Let a and b be nonadjacent vertices. If S is a minimal a,b-separator which contains only neighbors of a, then S is called* close to a.

LEMMA 4. *If a and b are nonadjacent, then there exists exactly one minimal a,b-separator close to a.*

Proof. Let S be a minimal a,b-separator close to a. For every vertex in S there is a path to b which does not use any other neighbors of a, since S is minimal. On the other hand, if x is a neighbor of a such that there is a path to b without any other neighbors of a, then x must be an element of S; otherwise, there is a path between x and b which avoids S and this is a contradiction since x is in the component of $G[V-S]$ that contains a. □

Notice that a minimal separator close to a can easily be computed as follows. Start with $S = N(a)$. Clearly, since a and b are nonadjacent, S separates a and b. Let C_b be the connected component of $G[V-S]$ containing b. Let $S' \subseteq S$ be the set of those vertices of S which have at least one neighbor in C_b. By Lemma 2, S' is a minimal a,b-separator, and since it only contains neighbors of a, it is close to a.

LEMMA 5. *Let S be a minimal a,b-separator close to a and let C_a and C_b be the connected components containing a and b, respectively. Let $S^* \neq S$ be another minimal a,b-separator. Then $S^* \subset S \cup C_b$.*

Proof. Since S^* is a minimal a,b-separator, $S^* \subset C_a \cup C_b \cup S$. Assume S^* has a vertex $x \in C_a$. $S^* \setminus \{x\}$ does not separate a and b; hence there is a path P between a and b using x but no other vertex of S^*. Since S is a minimal separator, P goes through a vertex $y \in S$. Since S is close to a, y is adjacent to a. Hence there is a path $P' \subset P$ between a and b that does not contain x. This is a contradiction since P' contains no vertex of S^*. □

In sections 3 and 4 we show how to obtain new minimal a,b-separators from a given one using so-called minimal pairs. A minimal pair is in some sense the smallest step to go from one minimal a,b-separator to the next one. The main difficulty is to prove that we indeed obtain all minimal separators by using small steps only.

In section 5 we describe an algorithm that computes all minimal a,b-separators for a given pair of nonadjacent vertices a and b in a breadth-first-search manner; we prove that it is correct and we analyze its time complexity. We end with some concluding remarks and open problems.

3. Good pairs. Let $G = (V, E)$ be a graph and let a and b nonadjacent vertices in G. Let S be a minimal a, b-separator and let C_a and C_b be the connected components containing a and b, respectively.

DEFINITION 6. *Let $\Delta \subseteq C_a \setminus \{a\}$ and let C'_a be the connected component of $G[C_a - \Delta]$ that contains a. Let $N \subseteq S$ be the set of vertices in S that do not have a neighbor in C'_a. The pair (Δ, N) is called* good *for S if the following conditions are satisfied:*

1. *$N \neq \emptyset$.*
2. *Each $\delta \in \Delta$ has at least one neighbor in C'_a.*
3. *Each $\delta \in \Delta$ either has a neighbor in N or there exists a vertex $x \in N$ and a connected component D of $G[C_a - \Delta]$ such that both x and δ have at least one neighbor in D.*

LEMMA 7. *If S is close to a, then there is no good pair.*

Proof. Assume (Δ, N) is a good pair. Hence $\Delta \subseteq C_a \setminus \{a\}$. Let C'_a be the connected component of $G[C_a - \Delta]$ that contains a. The set N is defined as the subset of S that does not contain any neighbor in C'_a. Then $N = \emptyset$ since S contains only neighbors of a, but by definition $N \neq \emptyset$. □

Theorem 8 shows that a good pair defines a new separator. In Theorem 9 we show that each minimal a, b-separator can be obtained by a good pair for the separator that is close to b. In section 4 we show that only a restricted type of good pairs, called minimal pairs, has to be considered.

THEOREM 8. *Let (Δ, N) be a good pair. Define $S^* = (S \cup \Delta) \setminus N$. Then S^* is a minimal a, b-separator.*

Proof. Let C'_a be the connected component of $G[C_a - \Delta]$ that contains a. Clearly, S^* separates a and b, since vertices of N do not have neighbors in C'_a. Let C'_b be the connected component of $G[V - S^*]$ that contains b. Notice that $C_b \subset C'_b$, and since each vertex of N has a neighbor in C_b, $N \subset C'_b$.

Each vertex of S^* has at least one neighbor in C'_a by definition of a good pair, and each vertex of $S^* \setminus \Delta$ has at least one neighbor in C'_b since it has at least one neighbor in C_b. The only thing left to show is that each vertex of Δ has a neighbor in C'_b. Let $\delta \in \Delta$. By definition, either δ has a neighbor in N (and hence in C'_b) or there is a vertex $x \in N$ and a connected component D of $G[C_a - \Delta]$ such that both δ and x have a neighbor in D. D is also connected in $G[V - S^*]$ and, since x has a neighbor in D, $D \subset C'_b$. □

THEOREM 9. *Assume S is close to b. Let $S^* \neq S$ be a minimal a, b-separator. There exists a good pair (Δ, N) such that $S^* = (S \cup \Delta) \setminus N$.*

Proof. Let C^*_a and C^*_b be the connected components of $G[V - S^*]$ containing a and b, respectively.

First notice that $S^* \subset C_a \cup C_b \cup S$, since S^* is minimal. Since S is close to b, by Lemma 5, $S^* \subset S \cup C_a$. Let $\Delta = S^* \cap C_a$ and $N = S \setminus S^*$. We show that (Δ, N) is a good pair.

Since $S^* \neq S$ and both are minimal a, b-separators then $N \neq \emptyset$.

Let C'_a be the connected component of $G[C_a - \Delta]$ containing a. We show that N is exactly the set of vertices in S which do not have a neighbor in C'_a. In order to do this we claim that $C'_a = C^*_a$. Since C'_a is a connected component of $G[V - (\Delta \cup S)]$ and since $S^* \subset \Delta \cup S$, $C'_a \subseteq C^*_a$. Now assume there is a vertex $x \in N$ which has a neighbor $y \in C'_a$. Since S is close to b, x is a neighbor of b. This is a contradiction since there would be a path between a and b which does not use any vertex of S^*.

This shows that $C'_a = C^*_a$. Since S^* is minimal, N is exactly the set of vertices in S that do not have a neighbor in C'_a, and every vertex of $\Delta \cup (S \setminus N)$ has at least one neighbor in C'_a.

To prove the last item, first notice that $N \subset C^*_b$ and that C^*_b contains exactly those connected components D of $G[C_a - \Delta]$ for which there is a vertex $y \in N$ which has a neighbor in D. Now let $\delta \in \Delta$. Since S^*_a is minimal, δ has a neighbor x in C^*_b. Since δ only has neighbors in $C_a \cup S$, x must be an element of N or of some component D of $G[C_a - \Delta]$. In this second case, there must also be a vertex $y \in N$ which has a neighbor in D. □

4. Minimal pairs. Again let $G = (V, E)$ be a graph and let a and b be nonadjacent vertices in G. Let S be a minimal a, b-separator and let C_a and C_b be the connected components of $G[V - S]$ containing a and b, respectively. In this section we show how to find some good pairs.

DEFINITION 10. *Let $x \in S$ be nonadjacent to a. Let $C_a(x)$ be the subgraph induced by $C_a \cup \{x\}$. Let Δ be the minimal x, a-separator in $C_a(x)$ close to x, and let C'_a be the connected component containing a. Now let N be the set of vertices of S which do not have a neighbor in C'_a. The pair (Δ, N) is called the* minimal pair *for S and x.*

LEMMA 11. *A minimal pair is good.*

Proof. Notice that $x \in N$; hence $N \neq \emptyset$.

Now, Δ is a minimal x, a-separator in $C_a(x)$ and hence every vertex of Δ has a neighbor in C'_a.

Finally, if $\delta \in \Delta$, then δ is adjacent to x since Δ is close to x. Hence each vertex of Δ has a neighbor in N. □

We want to prove that we can find every minimal a, b-separator by starting with the minimal a, b-separator that is close to b and by recursively using minimal pairs. The following technical lemma proves this.

LEMMA 12. *Let (Δ, N) be a good pair for S. Let $x \in N$ and let (Δ^*, N^*) be the minimal pair for S and x. Let $S^* = (S \cup \Delta^*) \setminus N^*$. Define $\overline{\Delta} = \Delta \setminus \Delta^*$ and $\overline{N} = (N \setminus N^*) \cup (\Delta^* \setminus \Delta)$. Then*

1. *if $\overline{N} = \emptyset$, then $(S \cup \Delta) \setminus N = S^*$, and*
2. *if $\overline{N} \neq \emptyset$, then $(\overline{\Delta}, \overline{N})$ is a good pair for S^* and $(S \cup \Delta) \setminus N = (S^* \cup \overline{\Delta}) \setminus \overline{N}$.*

Proof. We start with some easy observations. Let C'_a be the connected component of $G[C_a - \Delta]$ that contains a and let C^*_a be the connected component of $G[C_a - \Delta^*]$ that contains a. Let $\Delta' = N(x) \cap \Delta$.

- $C'_a \subseteq C^*_a$ since Δ^* contains no vertices of C'_a.
- $\Delta' \subseteq \Delta^*$ since every vertex of Δ' has a neighbor in C'_a.
- $\Delta \setminus \Delta' \subseteq C^*_a$ since every vertex of Δ has a neighbor in C'_a.
- $N^* \subseteq N$ since $C'_a \subseteq C^*_a$.
- C'_a is exactly the connected component of $G[C^*_a - (\Delta \setminus \Delta')]$ containing a since $C^*_a - (\Delta \setminus \Delta')$ contains all vertices of C'_a but no vertex of Δ.
- The set of vertices in S^* without a neighbor in C'_a is exactly $\overline{N}$, which is easy to check.

Assume $\overline{N} = \emptyset$. Then $\Delta^* \subseteq \Delta$ and $N = N^*$ (since $N^* \subseteq N$). Now it is also clear that $\Delta^* = \Delta$; otherwise S^* and $(S \cup \Delta) \setminus N$ are two minimal a, b-separators of which one is properly contained in the other—which is impossible. Hence $S^* = (S \cup \Delta) \setminus N$.

Now assume $\overline{N} \neq \emptyset$. We show that $(\overline{\Delta}, \overline{N})$ is good for S^*. Notice that every vertex of $\overline{\Delta}$ has a neighbor in C'_a, since this holds for every vertex of Δ.

Let $\delta \in \overline{\Delta}$ and assume that δ has no neighbors in $\overline{N}$. Since $\delta \in C_a^*$, δ has no neighbor in N^*. Hence δ has no neighbor in N. Now (Δ, N) is a good pair; hence there is a vertex $z \in N$ and a connected component D of $G[C_a - \Delta]$ such that δ and z have a neighbor in D. Assume by way of contradiction that for no vertex of $\overline{N}$ there is a connected component in $G[C_a^* - \overline{\Delta}]$ such that both this vertex and δ have a neighbor in this component. The following observations lead to a contradiction.

- $N(\delta) \cap D \subseteq C_a^*$. Otherwise, since $\Delta^* \setminus \Delta' \subset \overline{N}$, δ has a neighbor in $\overline{N}$.
- $G[D - \Delta^*]$ is connected. Since otherwise every connected component has a vertex with a neighbor in $\Delta^* \setminus \Delta$, and hence there is a connected component and some vertex in $\overline{N}$ such that both this vertex and δ have a neighbor in this component.
- D contains no vertices of Δ^* by the same argument.

This shows that $D \subset C_a^*$. If $z \in N^*$ then z can have no neighbors in D, since z has no neighbors in C_a^*. Hence $z \in N \setminus N^*$. This is a contradiction, since now there is a connected component D in $G[C_a^* - \overline{\Delta}]$ and a vertex $z \in \overline{N}$ such that both z and δ have a neighbor in D.

The fact that $(S \cup \Delta) \setminus N = (S^* \cup \overline{\Delta}) \setminus \overline{N}$ is obvious. □

Starting with the minimal separator close to b, Theorem 9 ensures that a good pair (Δ, N) exists for every minimal separator. If, at one point, we arrived at a minimal separator S, Lemma 12 shows that we can find the minimal separator $(S \cup \Delta) \setminus N$ by successively choosing minimal pairs (Δ^*, N^*). Notice that the component of S^* containing a is smaller than the component of S containing a. Hence, after a finite number of steps we reach $(S \cup \Delta) \setminus N$. Consequently we obtain the following result.

COROLLARY 13. *Let S be a minimal a,b-separator and let S_1 be the minimal a,b-separator close to b. There exists a sequence $(\Delta_1, N_1), \ldots, (\Delta_t, N_t)$ such that*

1. *(Δ_1, N_1) is a minimal pair for S_1 and some vertex $x_1 \in N_1$.*
2. *for $i = 2, \ldots, t$, (Δ_i, N_i) is a minimal pair for $S_i = (S_{i-1} \cup \Delta_{i-1}) \setminus N_{i-1}$ and some vertex $x_i \in N_i$.*
3. *for $i = 1, \ldots, t$, Δ_i and a are in the same connected component of $G[V - S_i]$.*
4. *$S = (S_t \cup \Delta_t) \setminus N_t$.*

5. An algorithm listing minimal separators. In this section we give an algorithm that, given a graph G and two nonadjacent vertices a and b, finds all minimal a,b-separators. This algorithm is displayed in Figure 1.

THEOREM 14. *Let S be the minimal a,b-separator that is close to b and let $\mathcal{T} = \{S\}$ and $\mathcal{Q} = \{S\}$. Then a call separators$(G, a, b, \mathcal{T}, \mathcal{Q})$ determines a set $\mathcal{Q}$ containing all minimal a,b-separators.*

Proof. By Corollary 13 the set $\mathcal{Q}$ contains all minimal a,b-separators. By Lemma 11 and Theorem 8 all sets in $\mathcal{Q}$ are minimal separators. □

Remark. If we let $\mathcal{T} = \{\{b\}\}$ and $\mathcal{Q} = \emptyset$, then a call separators$(G, a, b, \mathcal{T}, \mathcal{Q})$ has the same result.

THEOREM 15. *Let $R(a,b)$ be the number of minimal a,b-separators (for nonadjacent vertices a and b). The algorithm to determine all minimal a,b-separators can be implemented to run in time $O(n^3 R(a,b))$.*

Proof. Assume that the graph is given with an adjacency matrix. The minimal separator S that is close to b can easily be found in $O(n^2)$ time as follows. Initialize $S = N(b)$. Determine the connected component C_a of $G[V - S]$. Remove vertices from S that do not have a neighbor in C_a.

Consider the time it takes to compute $\mathcal{T}'$. For each $S \in \mathcal{T}$ and for each $x \in S$ not adjacent to a we have to do the following computations. Determining Δ takes at

```
procedure separators(G, a, b, 𝒯, 𝒬)
input: Graph G and non adjacent vertices a and b and
       sets 𝒯 and 𝒮 of minimal a,b-separators.
output: Set 𝒬 of all minimal a,b-separators in G.
begin
    𝒯' := ∅;
    for each S ∈ 𝒯 do
    begin
        Determine C_a;
        { C_a is the connected component of G[V − S] that contains a.}
        for each x ∈ S which is not adjacent to a do
        begin
            Determine Δ;
            {Δ is the minimal x,a-separator in C_a(x) that is close to x.}
            Determine C'_a;
            { C'_a is the connected component of G[C_a − Δ] that contains a.}
            Determine N;
            {N is the set of vertices in S that do not have a neighbor in C'_a,}
            S* := (S ∪ Δ) \ N;
            T' := T' ∪ {S*}
            {Add S* to T' only if not yet present!}
        end for
    end for;
    𝒬 := 𝒬 ∪ 𝒯';
    separators(G, a, b, 𝒯', 𝒬)
end.
```

FIG. 1. *Algorithm listing minimal separators.*

most $O(n^2)$ time. Computing C'_a and N can clearly be done in $O(n^2)$ time. Hence the time to compute $\mathcal{T}'$ (which may contain elements that are already in $\mathcal{Q}$) can be performed in $O(n^3|\mathcal{T}|)$ time.

We have to remove minimal separators that have been found before from the new set $\mathcal{T}'$. We can do this by keeping $\mathcal{Q}$ in a suitable data structure, allowing an update in $O(n|\mathcal{T}'|\log R(a,b)) = O(n^2|\mathcal{T}'|) = O(n^3|\mathcal{T}|)$ time. It follows that the computation of $\mathcal{T}'$, containing only new minimal separators, can be performed in $O(n^3|\mathcal{T}|)$ time. Since each newly computed set $\mathcal{T}'$ contains only minimal separators that have not been found before, it follows that the total time needed by the algorithm is $O(n^3R(a,b))$. □

COROLLARY 16. *The set of all minimal separators of a graph can be found in $O(n^5R)$ time, where n is the number of vertices in the graph and R is the total number of minimal separators.*

A somewhat different result is the following.

THEOREM 17. *Assume G has at least R minimal separators. There exists an algorithm that finds R different minimal separators in $O(n^5R)$ time.*

Proof. The claimed algorithm is simply the following. The algorithm listing minimal separators described above is stopped when R different ones have been found. It may take time at most $O(n^5R)$, trying different pairs of nonadjacent vertices for which the total number of different minimal separators is smaller than R. Assume a pair of vertices has been found with enough new minimal separators. Now the analysis in the proof of Theorem 15 shows the claimed result. □

6. Conclusions. In this paper we have presented an algorithm to determine a list of all minimal vertex separators of a graph. The algorithm needs only polynomial time per separator that is found. We would like to mention some open problems.

First of all, we feel that it should be possible to improve the running time of the algorithm presented here.

A related concept is that of an *inclusion minimal separator*. This is a minimal separator with the additional constraint that no proper subset is also a minimal separator. The following lemma shows that our algorithm can be used to find all inclusion minimal separators. However, the example given in the introduction illustrates that this may not be an efficient way to do this.

LEMMA 18. *A separator S of a graph $G = (V, E)$ is inclusion minimal if and only if every vertex of S has a neighbor in every connected component of $G[V - S]$.*

It follows that a list of all inclusion minimal separators can easily be obtained from the list of all minimal separators. Until now, we have not been able to find an efficient algorithm which finds all inclusion minimal separators.

Acknowledgments. We thank B. Monien (University of Paderborn, Germany) for drawing our attention to this important problem. We are grateful to an anonymous referee for helpful comments.

REFERENCES

[1] S. ARNBORG, *Efficient algorithms for combinatorial problems on graphs with bounded decomposability—A survey*, BIT, 25 (1985), pp. 2–23.
[2] S. ARNBORG, J. LAGERGREN, AND D. SEESE, *Easy problems for tree-decomposable graphs*, J. Algorithms, 12 (1991), pp. 308–340.
[3] S. ARNBORG AND A. PROSKUROWSKI, *Linear time algorithms for NP-hard problems restricted to partial k-trees*, Discrete Appl. Math., 23 (1989), pp. 305–314.
[4] H. BODLAENDER, *A tourist guide through treewidth*, Acta Cybernet., 11 (1993), pp. 1–21.
[5] H. BODLAENDER, *A linear time algorithm for finding tree-decompositions of small treewidth*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, ACM, New York, 1993, pp. 226–234.
[6] H. BODLAENDER, T. KLOKS, AND D. KRATSCH, *Treewidth and pathwidth of permutation graphs*, SIAM J. Discrete Math., 8 (1995), pp. 606–616.
[7] H. BODLAENDER, T. KLOKS, D. KRATSCH, AND H. MÜLLER, *Treewidth and Minimum Fill-in on d-Trapezoid Graphs*, Technical Report UU-CS-1995-34, Utrecht University, The Netherlands, 1995.
[8] F. GAVRIL, *Algorithms on clique separable graphs*, Discrete Math., 19 (1977), pp. 159–165.
[9] L. A. GOLDBERG, *Efficient Algorithms for Listing Combinatorial Structures*, Cambridge University Press, Cambridge, 1993.
[10] M. C. GOLUMBIC, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
[11] A. KANEVSKY, *On the number of minimum size separating vertex sets in a graph and how to find all of them*, in Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, PA, 1990, pp. 411–421.
[12] T. KLOKS, *Treewidth—Computations and Approximations*, Lecture Notes in Computer Science 842, Springer-Verlag, Berlin, 1994.
[13] T. KLOKS, *Treewidth of circle graphs*, Internat. J. Found. Comput. Sci., 7 (1996), pp. 111–120.
[14] T. KLOKS, *Minimum Fill-in for Chordal Bipatite Graphs*, Technical Report RUU-CS-93-11, Department of Computer Science, Utrecht University, The Netherlands, 1993.
[15] T. KLOKS AND D. KRATSCH, *Treewidth of chordal bipartite graphs*, J. Algorithms, 19 (1995), pp. 266–281.
[16] T. KLOKS, D. KRATSCH, AND J. SPINRAD, *On treewidth and minimum fill-in of asteroidal triple-free graphs*, Theoret. Comput. Sci., 175 (1997), pp. 309–335.
[17] D. KRATSCH, *The Structure of Graphs and the Design of Efficient Algorithms*, Habilitation thesis, F.-Schiller-Universität, Jena, Germany, 1996.

[18] A. Parra and P. Scheffler, *How to use the minimal separators of a graph for its chordal triangulation*, in Proceedings of the 22rd International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science 944, Springer-Verlag, Berlin, 1995, pp. 123–134.
[19] R. E. Tarjan, *Decomposition by clique separators*, Discrete Math., 55 (1985), pp. 221–232.
[20] S. H. Whitesides, *An algorithm for finding clique cut-sets*, Inform. Process. Lett., 12 (1981), pp. 31–32.

SIAM J. COMPUT.
Vol. 27, No. 3, pp. 614–636, June 1998
© 1998 Society for Industrial and Applied Mathematics
002

EFFICIENT SELF-EMBEDDING OF BUTTERFLY NETWORKS WITH RANDOM FAULTS*

HISAO TAMAKI†

Abstract. We study the embedding of the butterfly network in its faulty version, where each node is independently faulty with some constant probability p. We give a method of such self-embedding of the N-node butterfly with $O(1)$ load, $O((\log\log N)^{2.6})$ dilation, and $O((\log\log N)^c)$ congestion, which succeeds with probability at least $1 - N^{-1}$ if $p < 1 - \sqrt{2/3} \simeq 0.1835$, where c is a constant that depends on p; c is about 8.84 for $p = 0.1$ and approaches $\log_2 40 \simeq 5.32$ as $p \to 0$. The method is constructive and in fact yields an $N \log^{O(1)} N$ time deterministic algorithm to construct the claimed embedding with the claimed success probability when given the random faulty butterfly. We also show that we can make the dilation as low as $O(\log\log N)$, although at the cost of $\log^{O(1)} N$ congestion. These embeddings are level-preserving in the sense that each node is mapped to a node in the same level of the butterfly as the original node. We also derive a lower bound of $\log\log N - o(\log\log N)$ on the dilation of a level-preserving embedding with N^α load, for any constant $\alpha < 1/3$ and any constant node-failure probability $p > 0$. Thus, the bounds on dilation are tight up to a constant factor, as far as level-preserving embeddings are concerned.

Key words. butterfly network, random faults, network embedding, interconnection networks, embedding

AMS subject classifications. 60C05, 68M07, 68Q22

PII. S0097539794270364

1. Introduction.

In this paper, we study the robustness of interconnection networks against random static faults. We assume that each node of the network is faulty with some constant probability, independently of other nodes. Faulty nodes are conceptually removed from the network together with all of its incident edges. The question we address is if the remaining subnetwork can efficiently embed the fault-free version of the network in the following sense. In general, an *embedding* of a network G in a network H maps each node of G to a node of H and each edge of G to a path of H so that, if an edge (u, v) of G is mapped to a path P of H, then nodes u and v are mapped to the endpoints of P. In our context of self-embedding, G is the fault-free network and H is the network after the removal of faulty nodes. The *load* of the embedding is the maximum number of nodes of G mapped to a single node of H, the *dilation* is the maximum length of a path of H to which some edge is mapped, and the *congestion* is the maximum number of edges of G whose image paths share a common edge of H. A self-embedding with small load, dilation, and congestion implies an efficient simulation of the fault-free network by the faulty network and thus can be considered as evidence of the robustness of the network structure. (In fact, the slowdown factor of the simulation is $O(load + dilation + congestion)$ due to the result of Leighton, Maggs, and Rao [12] on off-line routing.)

Under this model, Håstad, Leighton, and Newman [6] show that the hypercube is highly robust, supporting a self-embedding with $O(1)$ load, $O(1)$ dilation, and $O(1)$ congestion, with high probability. Aiello and Leighton [1] give another proof of this result as an application of their more general embedding technique. This result heavily

*Received by the editors June 27, 1994; accepted for publication (in revised form) February 7, 1996.

http://www.siam.org/journals/sicomp/27-3/27036.html

†IBM Tokyo Research Laboratory, 1623-14 Shimotsuruma, Yamato, 242 Japan (htamaki@trl.ibm.co.jp). This work was done while the author was at the University of Toronto.

depends on the logarithmic degree of the hypercube, and one naturally asks if a similar robustness can be achieved by bounded-degree networks. For the two-dimensional array, this question is resolved by Kaklamanis et al. [7], who show the upper bound of $O(1)$ load, $O(\log N)$ dilation, and $O(1)$ congestion (with high probability), together with a matching lower bound that any $O(1)$-load embedding requires $\Omega(\log N)$ dilation with high probability. Here, and throughout this paper, we use N to denote the number of nodes in the network. Kaklamanis et al. also give almost tight bounds for the three-dimensional array. Tamaki [16] introduces a new bounded-degree network called cube-connected arrays which admits $O(1)$ load self-embedding with dilation and congestion both $(\log\log N)^{1+o(1)}$. However, the problem has remained open for the so-called hypercubic networks, such as the butterfly, the cube-connected-cycles (CCC), the shuffle-exchange, and the de Bruijn networks. Annexstein [3] gives self-embeddings with $O(\log\log N)$ dilation and $O(1)$ congestion for these networks, but under a weaker model in which an edge is allowed to be mapped to a path containing faulty nodes. Karlin and Nelson [8] show that a faulty butterfly contains a linear-sized connected component with high probability. This means that $O(1)$-load embedding is possible, but their proof technique does not seem to lead to good bounds for dilation and congestion.

Recently, Leighton, Maggs, and Sitaraman [11] extensively studied worst-case faults in bounded-degree networks, and as one of the consequences, interestingly, derived a result on the butterfly with *random faults*: a faulty butterfly can simulate the complete butterfly with slowdown factor $2^{O(\log^* N)}$, which is very close to a constant. This simulation is not based on embedding but uses the more involved scheme of replication [10].

In this paper, we give an efficient self-embedding of the butterfly network with random faults.

THEOREM 1.1. *For node-failure probability $p < 1 - \sqrt{2/3} \simeq 0.1835$, a faulty N-node butterfly can embed a fault-free N-node butterfly with $O(1)$ load, $O((\log\log N)^{2.6})$ dilation, and $O((\log\log N)^c)$ congestion, with probability at least $1 - N^{-1}$, where c is a constant that depends on p; c is about 8.84 for $p = 0.1$ and approaches $\log 40 \simeq 5.32$[1] as $p \to 0$.*

The proof is constructive in the sense that we can extract an efficient deterministic algorithm (that runs in $N \log^{O(1)} N$ steps) for constructing the embedding.

We also show that if we allow congestion of $\log^{O(1)} N$, then we can make the dilation of the embedding as small as $O(\log\log N)$.

Following [11], we call a self-embedding of the butterfly *level-preserving* if it maps each node of the butterfly to a node in the same level as the original. Both of our embeddings in the above results are level-preserving. We also prove the following matching lower bound on the dilation of level-preserving self-embeddings.

THEOREM 1.2. *For any positive constants p and $\alpha < 1/3$, the probability is $1 - 2^{-N^{\Omega(1)}}$ that any level-preserving self-embedding with load N^α in a faulty N-node butterfly with node failure probability p requires dilation at least $\log\log N - o(\log\log N)$.*

The work of Leighton, Maggs, and Sitaraman [11] includes a negative result for worst-case faults: to ensure a self-embedding with $O(1)$ load and dilation, only poly-logarithmic number of worst case faults can be allowed in a butterfly. Their technique does not seem to apply to random faults, nor does ours to worst-case faults.

[1] Throughout the paper, all logarithms are base 2.

This paper improves on its preliminary version [15] in three major points: (1) we give an explicit allowable value of the node-failure probability p for our embedding; (2) we include a dilation upper bound of $O(\log\log N)$; and (3) we have a matching lower bound on the dilation of level-preserving embeddings, improving on the weaker lower bound of $\log\log\log N$ in the preliminary version.

The rest of the paper is organized as follows. In section 2, we define some necessary notions and introduce notations. In section 3, we show that a faulty butterfly maintains a large connected component and a reasonably good routing capability with very high probability. In section 4, we improve the congestion of the routing of section 3 at a slight expense of dilation. In section 5, this result of section 4 is applied to small subbutterflies of a faulty butterfly and yields our main embedding result. On the other hand, the result of section 3, similarly applied, yields the embedding with better dilation and exponentially worse congestion. In section 6, we prove the dilation lower bound. Finally, in section 7 we discuss some open problems.

2. Preliminaries. In this section, we define the butterfly network and review some of its structural properties. We review some probabilistic tools we use and also introduce some terminology for routing.

For two integers m and n, we denote by $[m,\ldots,n]$ the set of integers i satisfying $m \le i \le n$. We also write $[n]$ for $[1,\ldots,n]$. Let m be a positive integer. The m-stage butterfly, denoted by $\mathcal{B}_m$, is an undirected graph on node set $[0,\ldots,m]\times\{0,1\}^m$. If v is a node of the form $\langle i,x\rangle$, $i\in[0,\ldots,m]$ and $x\in\{0,1\}^m$, we call i the *level index* and x the *row index* of v. Two nodes $\langle i,x\rangle$ and $\langle j,y\rangle$ are adjacent if and only if (1) $|i-j|=1$, and (2) x and y are equal at all bit positions except possibly the kth, where k is the larger of i and j. For a fixed i, we refer to the set of all nodes with level index i as *level* i and to the set of all edges between level $i-1$ and level i as *stage* i. We call an edge $(\langle i-1,x\rangle,\langle i,y\rangle)$ a *straight edge* if $x=y$; a *cross edge* if x and y do differ at the ith bit position. We also refer to the set of all nodes with row index x, for a fixed x, as *row* x. Note that if we collapse each row x into a single node, $\mathcal{B}_m$ degenerates into an m-dimensional hypercube, with a cross edge in stage i becoming an edge in dimension i. As in Figure 1, our convention is to draw, and regard, the lower levels to be on the left-hand side.

The following notions and facts about the butterfly will be frequently used. We call a path in $\mathcal{B}_m$ *monotone* if it visits each level at most once. If the leftmost node of a monotone path is u and the rightmost node is v, then we say that the monotone path *starts at* u and *ends at* v. It is sometimes convenient to see a monotone path as being specified by the starting node and a binary string. Let us label each edge $(\langle i-1,x\rangle,\langle i,y\rangle)$ of $\mathcal{B}_m$ with the ith bit of y. When the butterfly is drawn as in Figure 1, of those two edges that go out to the right from each node, the upper one is labeled with 0 and the lower one is labeled with 1. The *label* of a monotone path $(v_0,v_1,\ldots,v_k)$, where the vertices are listed from left to right, is defined to be a bit string $b_1,\ldots,b_k$, where each b_i, $i\in[k]$ is the label of the edge (v_{i-1},v_i). The following facts about $\mathcal{B}_m$ can be easily verified. For $x\in\{0,1\}^*$, $|x|$ denotes the length of x.

1. A node v on level i and a binary string w, $|w|\le m-i$, uniquely specify a monotone path that starts at v and has w as its label.
2. If a monotone path starts at node $\langle i,x\rangle$ and has label w, then it ends at node $\langle i+|w|,ywz\rangle$, where y is the prefix of x with length i and z is the suffix of x with length $m-|w|-i$.
3. For each pair of nodes u on level 0 and v on level m, there is a unique monotone path in $\mathcal{B}_m$ that starts at u and ends at v. The label of this unique monotone path is the row index of v.

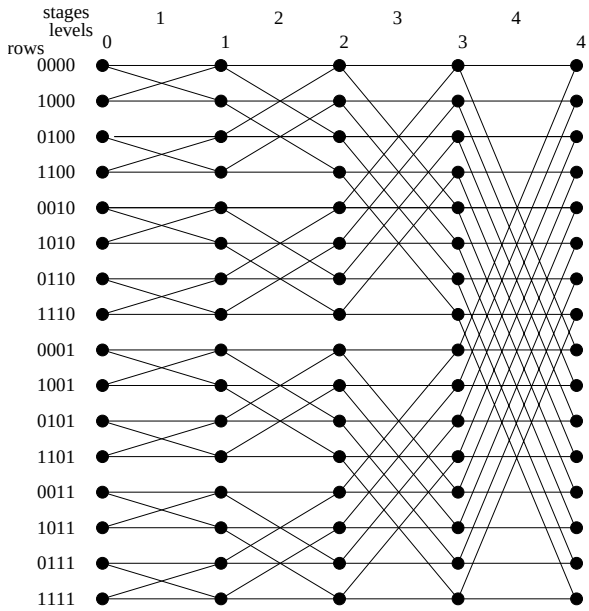


FIG. 1. *The 4-stage butterfly.*

4. For each node u on level 0, the graph formed by all the monotone paths of length m starting at u is a complete binary tree rooted at u.
5. For any monotone path $(v_0, v_1, \ldots, v_m)$ from level 0 to level m, there is an automorphism of $\mathcal{B}_m$ that maps each v_i to node $\langle i, 0^m \rangle$.

Let k and d be integers such that $0 \le k \le k + d \le m$. For $x \in \{0,1\}^k$ and $y \in \{0,1\}^{m-k-d}$, the subgraph of $\mathcal{B}_m$ induced by the set of nodes $\{\langle k + l, xwy \rangle \mid 0 \le l \le d, w \in \{0,1\}^d\}$ is isomorphic to $\mathcal{B}_d$; the standard isomorphism maps node $\langle k + l, xwy \rangle$ of $\mathcal{B}_m$ to node $\langle l, w \rangle$ of $\mathcal{B}_d$. We call this subgraph a *d-stage subbutterfly of $\mathcal{B}_m$ starting at level k (and ending at level $k + d$)*. When we refer to a subbutterfly, we usually view the graph through the standard isomorphism above. If B is a subbutterfly of $\mathcal{B}_m$, *Left*(B) will denote the set of nodes of B on its leftmost (lowest) level and *Right*(B) will denote the set of nodes on its rightmost (highest) level. In particular, *Left*$(\mathcal{B}_m)$ and *Right*$(\mathcal{B}_m)$ are, respectively, the set of nodes on levels 0 and m.

Let F be a subset of $V(\mathcal{B}_m)$. We write $\mathcal{B}_m\langle F \rangle$ to denote the subgraph of $\mathcal{B}_m$ induced by the node set $V(\mathcal{B}_m) \setminus F$. Informally, we will view $\mathcal{B}_m\langle F \rangle$ as the *faulty butterfly with fault set F*. For a subgraph B of $\mathcal{B}_m$, we extend this notation by defining $B\langle F \rangle$ to be the intersection of B and $\mathcal{B}_m\langle F \rangle$. We are primarily interested in the case where F is a random subset of $V(\mathcal{B}_m)$. Let S be a finite set and let X be a random subset of S such that each element of S is independently included in X with probability p. In other words, each fixed X is assigned probability $p^{|X|}(1-p)^{|S|-|X|}$. We denote this probability distribution by $\mathbf{randsub}(S, p)$ and write $X \in \mathbf{randsub}(S, p)$ to mean that X is chosen at random according to this distribution. Then $\mathcal{B}_m\langle F \rangle$, where $F \in \mathbf{randsub}(V(\mathcal{B}_m), p)$, induces a probability distribution over the subgraphs of $\mathcal{B}_m$. We denote this probability distribution by $\mathcal{B}_m\langle p \rangle$ and call it *the randomly faulty m-stage butterfly with node-failure probability p*.

We will use the following forms of the bound on the probability of a large deviation of a sum of independent random variables, usually called the *Chernoff bound*. The proofs can be found in the standard textbooks such as Alon and Spencer [2] or

Motwani and Raghavan [13]. Let X be a sum of independent random variables, each of which takes either 0 or 1 as its value. Let $\mu = \mathbf{E}(X)$ be the expected value of X.

PROPOSITION 2.1. *Let $\epsilon > 0$. Then, $P(|X - \mu| > \epsilon\mu) < e^{-c_\epsilon \mu}$, where c_ϵ is a positive constant depending only on ϵ.*

In one occasion, where we want to bound the probability that X deviates from its mean by more than a constant fraction, we will use the following.

PROPOSITION 2.2. *For any $T > \mu$, $P(X > T) < (\frac{e\mu}{T})^T$.*

There will be a few occasions where we need to deal with a sum of random variables which are not totally independent. If those random variables can be grouped so that the random variables in a single group are mutually independent, then we can apply the Chernoff bound to each subset. This reasoning gives the following proposition which we will refer to as the *partition Chernoff bound*.

PROPOSITION 2.3. *Let $\epsilon > 0$ be a constant and let $0 < p < 1$ be a value that possibly depends on n. Let X_i, for each $i \in [n]$, be a random variable which assumes value 1 with probability p and 0 with probability $1 - p$. Let $X = \sum_{i\in[n]} X_i$. Let $f(n)$ and $g(n)$ be functions such that $\log f(n) = o(pg(n))$, and suppose the set $[n]$ can be partitioned into $f(n)$ subsets with at least $g(n)$ elements each so that, for each part S of the partition, random variables X_i, $i \in S$ are mutually independent. Then, $P(|X - pn| > \epsilon pn) = 2^{-\Omega(pg(n))}$.*

Proof. For $|X - pn| > \epsilon pn$ to happen, there must be some part S of the partition such that $|\sum_{i\in S} X_i - p|S|| > \epsilon p|S|$. The probability of this event for a fixed S is at most $2^{-\Omega(pg(n))}$ by Proposition 2.1. Therefore, the probability that this happens for any S in the partition is at most $f(n)2^{-\Omega(pg(n))} = 2^{-\Omega(pg(n))}$. □

Now we turn to some terminology for routing. A *routing request* on a graph G is a multiset of ordered pairs of nodes of G. A routing request is *one-to-one* if each node appears at most once as the first component of a pair and at most once as the second component of a pair in the request. Otherwise it is *many-to-many*. A routing request on a butterfly is *end-to-end* if each pair consists of a node at level 0 and a node at the last level. A multiset Π of paths of G *realizes* a routing request R if it provides each pair in the request with a path connecting the pair; more formally, if there is a one-to-one mapping f from R to Π such that $f(u, v)$ for every $(u, v) \in R$ is a path with endpoints u and v. The *congestion* of a multiset of paths Π is the maximum number of times a single edge of G appears in the paths of Π. Let U and V be some sets of nodes of G. We say the pair (U, V) is (λ, γ)-*routable* in G if every one-to-one routing request $R \subseteq U \times V$ can be realized by a multiset of paths with maximum length λ and congestion γ. We say U is (λ, γ)-routable in G if (U, U) is (λ, γ)-routable in G.

Let $Q_m = \{0, 1\}^m$ denote the set of nodes of the m-cube and let $Q_m(i, b)$, $i \in [m]$ and $b \in \{0, 1\}$ denote the set of nodes of the subcube of the m-cube where the ith coordinate is fixed to b. For $0 < \delta < 1$, we say that a subset U of Q_m is δ-*dense* if at least a $(1 - \delta)$-fraction of each $Q_m(i, b)$ is contained in U, i.e., $|U \cap Q_m(i, b)| \geq (1 - \delta)2^{m-1}$ for every $i \in [m]$ and $b \in \{0, 1\}$. We will apply this notion to a subset U of a single level of $\mathcal{B}_m$, viewing this level as the set of nodes of the m-cube in the obvious manner.

3. $(O(m), O(2^m))$-routing in the faulty butterfly. The goal of this section is to show that, for appropriate values of δ and p, $\mathcal{B}_m\langle p\rangle$ almost surely contains a δ-dense subset of level 0 that is $(O(m), O(2^m))$-routable. This fact will later be applied to small subbutterflies of $\mathcal{B}_n$ with $m = O(\log n)$ to provide small dilation paths in the embedding of $\mathcal{B}_n$ into $\mathcal{B}_n\langle p\rangle$.

Let $\theta_p = (1-p)^2(1-\frac{p^2(2-p)^2}{(1-p)^4})$.

THEOREM 3.1. *Let δ, $0 < \delta < 1$ be given and fix p so that $\theta_p > 1-\delta$. Then, with probability at least $1 - 2^{-2^{\Omega(m)}}$, $\mathcal{B}_m\langle p\rangle$ has a δ-dense subset of level 0 that is $(4m, 2^{m+2})$-routable.*

In our later application, we need to take $\delta < 1/2$ and hence $\theta_p > 1/2$. To achieve this, it suffices to choose $p < 1 - \sqrt{2/3} \simeq 0.1835$.

The remainder of this section is devoted to the proof of Theorem 3.1. Let $\delta > 0$ be fixed. We first define a mapping Φ_m which is intended to map a faulty butterfly $\mathcal{B}_m\langle F\rangle$ to a set of nodes $U \subseteq \mathit{Left}(\mathcal{B}_m\langle F\rangle)$ such that U is δ-dense and $(O(m), O(2^m))$-routable. We will later prove that Φ_m succeeds with high probability in satisfying these conditions when applied to $\mathcal{B}_m\langle p\rangle$, provided $\theta_p > 1-\delta$.

We need some auxiliary definitions first. Let u and v be nodes on level 0 of $\mathcal{B}_m$ and let $\mathit{meet}(u,v)$ denote the smallest integer k such that u and v belong to the same k-stage subbutterfly starting at level 0. Let $F \subseteq V(\mathcal{B}_n)$ be a fault set. We say u and v are *confluent in* $\mathcal{B}_m\langle F\rangle$ if $\mathcal{B}_m\langle F\rangle$ contains two monotone paths of length $\mathit{meet}(u,v)$ that start at u and v and end at a common node. By convention, a node in $\mathit{Left}(\mathcal{B}_m\langle F\rangle)$ is confluent with itself. For each $v \in \mathit{Left}(\mathcal{B}_m\langle F\rangle)$, let $\mathit{Conf}_F(v)$ denote the set of all nodes that are confluent in $\mathcal{B}_m\langle F\rangle$ with v.

Now we define Φ_m. If there exists a node v such that $\mathit{Conf}_F(v)$ is δ-dense, then we set $\Phi_m(\mathcal{B}_m\langle F\rangle) = \mathit{Conf}_F(v_0)$, where v_0 is the first node in the natural ordering of rows such that $\mathit{Conf}_F(v_0)$ is δ-dense; otherwise we set $\Phi_m(\mathcal{B}_m\langle F\rangle) = \emptyset$.

It is clear that $\Phi_m(\mathcal{B}_m\langle F\rangle)$ is either δ-dense or empty; we say that Φ_m *succeeds* on $\mathcal{B}_m\langle F\rangle$ in the first case and that it *fails* in the second case. The following lemma shows that Φ_m provides what is required, whenever it succeeds.

LEMMA 3.2. *$\Phi_m(\mathcal{B}_m\langle F\rangle)$ is $(4m, 2^{m+2})$-routable in $\mathcal{B}_m\langle F\rangle$, when nonempty.*

Proof. Suppose a one-to-one routing request $Q \subseteq U \times U$ is given where $U = \mathit{Conf}_F(v_0) \neq \emptyset$. Given a pair $(u,v) \in Q$, we arbitrarily choose two monotone paths that witness the confluence of u with v_0 and two monotone paths that witness the confluence of v with v_0. Then, to the pair (u,v), we assign the path consisting of these four paths concatenated together. The total length of the path is at most $4m$. The congestion is at most 2^{m+2} because the number of monotone paths involved is $4|Q|$. □

To estimate the probability that Φ_m succeeds on $\mathcal{B}_m\langle p\rangle$, the following result from the theory of branching processes is needed. We define a *binary branching process with birth probability q* as the following special case of the Galton–Watson process. We start from a single organism in generation 0, which gives birth to some organisms in generation 1, which in turn have some children in generation 2, and so on. Each organism has exactly two potential children, each of which is actually born independently of each other and of any other organisms in the process (except for the obvious dependence to the birth of its ancestors), with probability q. Thus the number of children of each organism obeys a simple binomial distribution with generating function $f(z) = (qz + 1 - q)^2$. Let $Z_0, Z_1, \ldots$, denote the number of organisms in generation 0, 1, ..., respectively. It is well known that if $E(Z_1) = 2q > 1$, there is a positive probability that the process will continue forever ($Z_k > 0$ for every $k \geq 0$); the probability of extinction $\xi_q = P(\exists k Z_k = 0)$ is given as the unique root of $f(z) = z$ in the range $0 < z < 1$. Thus $\xi_q = ((1-q)/q)^2$ in our binary branching process. (See [4].) The following lemma due to Pippenger [14] states that the probability of Z_k being much smaller than its expected value $(2q)^k$ is almost as small as the extinction probability. More informally, a branching process is likely to thrive when it survives.

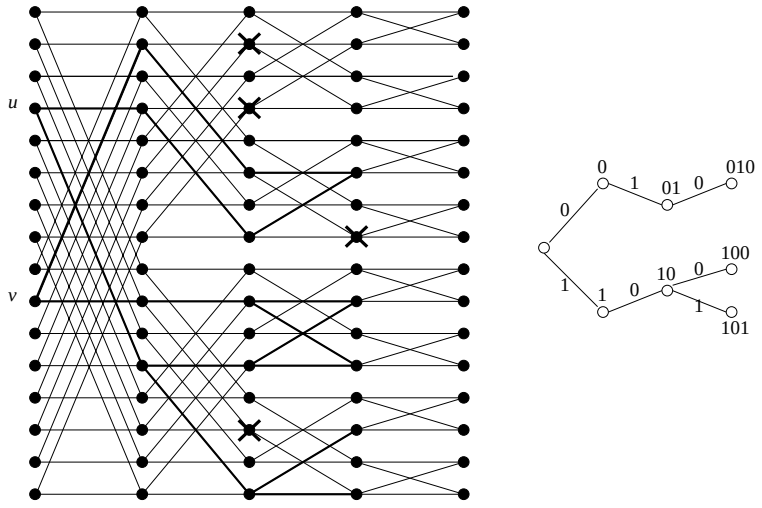


FIG. 2. *A confluence tree (rows are permuted from the standard drawing).*

LEMMA 3.3. *Suppose* $2q > 1$ *and let* τ *be an arbitrary constant* $0 < \tau < 1$. *Then* $P(Z_k < (2q)^{\tau k}) = \xi_q + o(1)$, *as* $k \to \infty$.

We use this lemma to estimate the probability that two nodes are confluent in $\mathcal{B}_m\langle p\rangle$. Let $F \subseteq V(\mathcal{B}_m)$ be a fault set. For each pair of nodes $u, v \in \mathit{Left}(\mathcal{B}_m)$, we define the *confluence tree of* u *and* v *with respect to* F, denoted by $T_{uv}(F)$, as follows. $T_{uv}(F)$ is a rooted binary tree in which each node is a binary string of length at most $\mathit{meet}(u, v)$. Each such string w is a node of $T_{uv}(F)$ if and only if both monotone paths of $\mathcal{B}_m$ with label w that starts at u and v are contained in $\mathcal{B}_m\langle F\rangle$. It is clear that the node set of $T_{uv}(F)$ is closed under the prefix operation; if ww' is a node of $T_{uv}(F)$, then so is w. The edge set of $T_{uv}(F)$ is defined so that there is an edge from w_1 to w_2 if and only if $w_2 = w_1 0$ or $w_2 = w_1 1$. Thus, the empty string is the root of $T_{uv}(F)$ and, in general, each string with length i, $i \in [0, \ldots, \mathit{meet}(u, v)]$ is at depth i of the tree if it is a node of $T_{uv}(F)$. Note that u and v are confluent in $\mathcal{B}_m\langle F\rangle$ if and only if their confluence tree has a node at depth $\mathit{meet}(u, v)$. See Figure 2.

When F is randomly chosen, $T_{uv}(F)$ gives rise to a binary branching process, to which we can apply the tools described above.

PROPOSITION 3.4. *For any two fixed nodes* $u, v \in \mathit{Left}(\mathcal{B}_m)$, *the probability that* u *and* v *are confluent in* $\mathcal{B}_m\langle p\rangle$ *is at least* $\theta_p = (1-p)^2(1 - \frac{p^2(2-p)^2}{(1-p)^4})$.

Proof. Fix $u, v \in \mathit{Left}(\mathcal{B}_m)$ and let $F \in \mathbf{randsub}(V(\mathcal{B}_m), p)$. We first observe that $T_{uv}(F)$ is essentially a binary branching process with birth probability $(1-p)^2$. To see this, consider a potential node w of $T_{uv}(F)$ with $|w| < \mathit{meet}(u, v)$ and its two potential children $w0$ and $w1$. Let u_0 be the node at which the monotone path with label $w0$ starting at u ends and define v_0 similarly with u replaced by v. If w is indeed a node of $T_{uv}(F)$, then $w0$ is a node of $T_{uv}(F)$ if and only if $u_0 \notin F$ and $v_0 \notin F$. Note that $u_0 = v_0$ if and only if $|w0| = \mathit{meet}(u, v)$. Therefore,

$$P(w0 \in T_{uv}(F) \mid w \in T_{uv}(F)) = \begin{cases} (1-p)^2 & \text{if } |w0| < \mathit{meet}(u, v) \\ 1-p & \text{if } |w0| = \mathit{meet}(u, v). \end{cases}$$

The same equality holds for the probability $P(w1 \in T_{uv}(F) \mid w \in T_{uv}(F))$ for the other potential child. Moreover, for any set W of potential nodes of $T_{uv}(F)$, events for

$w \in W$ and $b \in \{0,1\}$ that $wb \in T_{uv}(F)$ are mutually independent when conditioned that $w \in T_{uv}(F)$ for all $w \in W$. Thus, we can view $T_{uv}(F)$ up to depth $meet(u,v)-1$ as an initial part of a binary branching process with birth probability $(1-p)^2$. Though our process deviates at the last depth from the binary branching process, the change is the increased birth probability. Therefore, the probability that there is a descendant in generation $meet(u,v)$ only increases. It follows that the probability of $T_{uv}(F)$ having a node at depth $meet(u,v)$ is at least the survival probability $(1-\xi_{(1-p)^2})$ of the branching process, provided that neither u nor v is in F. The probability that neither u nor v is in F is $(1-p)^2$. Therefore the probability that u and v are confluent in $\mathcal{B}_m\langle F\rangle$ is at least $(1-p)^2(1-\xi_{(1-p)^2}) = \theta_p$. □

If the confluence of pairs were mutually independent, then we would be done; we would take p small enough so that θ_p is greater than $1-\delta$ and then apply the Chernoff bound to conclude that $Conf_F(v)$ is δ-dense with high probability for any $v \in Left(\mathcal{B}_m)$. We cope with the dependency in the following way.

Fix a positive constant $\tau < 1$ and let α be a constant that satisfies

$$0 < \alpha/(1-\alpha) < \tau(1+2\log(1-p))/(1-2\log(1-p)).$$

We assume $\log(1-p) > -1/2$, or equivalently $p < 1-\sqrt{2}/2$, so that the above range for $\alpha/(1-\alpha)$ is nonempty. Set $s = \lceil (1-\alpha)m \rceil$. In the following reasoning, we view $\mathcal{B}_m$ as split at level s and work with s-stage subbutterflies L_y, $y \in \{0,1\}^{m-s}$, starting at level 0 and $m-s$-stage subbutterflies R_x, $x \in \{0,1\}^s$, starting at level s. More precisely, L_y consists of the nodes $\{\langle i, xy\rangle \mid x \in \{0,1\}^s, i \in [0,\ldots,s]\}$ and R_x consists of the nodes $\{\langle i, xy\rangle \mid y \in \{0,1\}^{m-s}, i \in [s,\ldots,m]\}$. We call each L_y a *left subbutterfly* and R_x a *right subbutterfly*. Observe that, for each pair of left and right subbutterflies L_y and R_x, there is a unique row of $\mathcal{B}_m$ that intersects both L_y and R_x, namely row xy. For each node $v \in Left(\mathcal{B}_m)$, let $y(v)$ denote the length $m-s$ suffix of the row index of v. Note that $y(u) = y(v)$ if and only if $meet(u,v) \le s$.

Let $F \in \mathbf{randsub}(V(\mathcal{B}_m), p)$. For each pair of nodes $u, v \in Left(\mathcal{B}_m)$ such that $meet(u,v) > s$, let E_{uv} denote the event that the confluence tree $T_{uv}(F)$ has at least $(2(1-p)^2)^{\tau s}$ nodes at depth s. Note that $(2(1-p)^2)^s$ is the expected number of generation s descendants in a binary branching process with birth probability $(1-p)^2$. Thus, informally, E_{uv} is the event that the branching process associated to the confluence tree of u and v thrives up to generation s.

LEMMA 3.5. *$P(E_{uv}) \ge \theta_p - o(1)$ as $m \to \infty$.*

Proof. Since $meet(u,v) > s$, assuming that $u, v \notin F$, we can view the confluence tree $T_{uv}(F)$ up to depth s as a binary branching process with birth probability $(1-p)^2$. Apply Lemma 3.3 to this branching process. □

We next show that event E_{uv} implies the confluence of u and v with high probability.

LEMMA 3.6. *For every $u, v \in Left(\mathcal{B}_m)$ such that $meet(u,v) > s$, the probability that u and v are confluent in $\mathcal{B}_m\langle F\rangle$ conditional on the event E_{uv} is at least $1-2^{-2^{\alpha m}}$.*

Proof. Fix $u, v \in Left(\mathcal{B}_m)$ such that $meet(u,v) > s$. Note that $meet(u,v) > s$ implies $L_{y(u)} \ne L_{y(v)}$. Let W be the set of nodes at depth s of $T_{uv}(F)$ and suppose that event E_{uv} occurs, i.e., $|W| \ge (2(1-p)^2)^{\tau s}$. Let $w \in W$; w is a binary string of length s such that the two monotone paths with label w starting at u and v both avoid F. Let u_w and v_w be the nodes at which these monotone paths end, respectively. Since the row indices of u_w and v_w both have the same prefix w, they are in the same right subbutterfly R_w. (See Figure 3.) The probability that u_w and v_w are confluent in $R_w\langle F\rangle$ is at least $(1-p)^{2(m-s)}$, considering only one pair of monotone paths

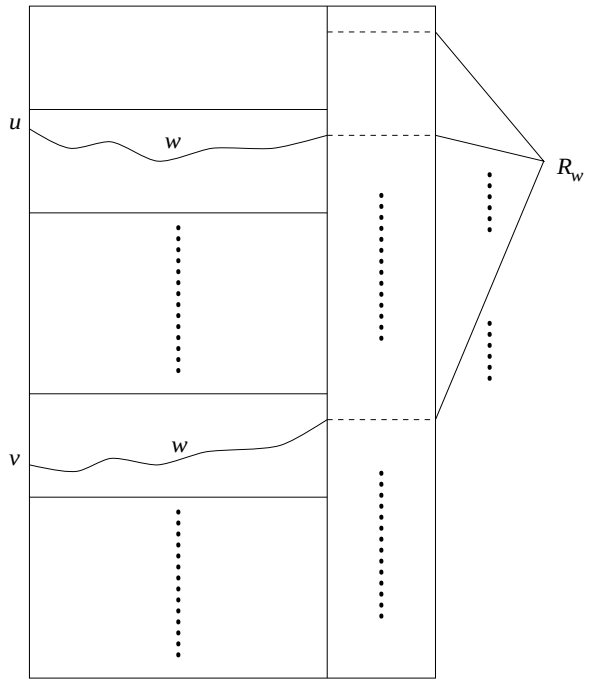


FIG. 3. *Confluent paths lead to a common right subbutterfly.*

that potentially connect these two nodes. Note here that conditioning on event E_{uv} does not affect the selection of faulty nodes at levels higher than s. Because this event that u_w and v_w are confluent in $R_w\langle F\rangle$ is independent for each w, and because $|W| \geq (2(1-p)^2)^{\tau s}$, the probability that none of these subbutterflies completes the confluence between u and v is at most

$$\begin{aligned}(1-(1-p)^{2(m-s)})^{(2(1-p)^2)^{\tau s}} &\leq e^{-(1-p)^{2(m-s)}(2(1-p)^2)^{\tau s}} \\ &\leq e^{-(1-p)^{2\alpha m}(2(1-p)^2)^{\tau(1-\alpha)m}} \\ &\leq e^{-2^{\alpha m}},\end{aligned}$$

where the last inequality follows from $(2(1-p)^2)^{\tau(1-\alpha)m} > (2/(1-p)^2)^{\alpha m}$ which is immediate from our choice of α. □

Let u be a node in $\mathit{Left}(\mathcal{B}_m)$ and let H be a fixed subgraph of $L_{y(u)}$. For each $v \in \mathit{Left}(\mathcal{B}_m)$ such that $y(v) \neq y(u)$, consider the conditional probability $P(E_{uv} \mid L_{y(u)}\langle F\rangle = H)$. Since this conditional event depends only on $L_{y(v)}\langle F\rangle$, this probability does not depend on the location of $L_{y(v)}$ in the entire butterfly. It does not depend on the location of v in $L_{y(v)}$ either, because for any node v' in $\mathit{Left}(L_{y(v)})$, there exists an automorphism of $L_{y(v)}$ that maps v to v' and maps each node in $\mathit{Right}(L_{y(v)})$ to itself; the above conditional probability is invariant under this automorphism. Therefore, this conditional probability does not depend on the choice of v as long as $y(v) \neq y(u)$. We say that a subgraph H of $L_{y(u)}$ is *η-good for u*, if this conditional probability $P(E_{uv} \mid L_{y(u)}\langle F\rangle = H)$ is at least η.

LEMMA 3.7. *For any positive constant $\eta < \theta_p$, the probability that $L_{y(u)}\langle F\rangle$ is η-good for u is at least $\zeta - o(1)$, where $\zeta = (\theta_p - \eta)/(1-\eta) > 0$.*

Proof. By Lemma 3.5, we have

$$P(E_{uv}) \geq \theta_p - o(1)$$

for every pair u, v such that $y(u) \neq y(v)$. On the other hand,

$$\begin{aligned} P(E_{uv}) &= \sum_{H \subseteq L_{y(u)}} P(E_{uv} \mid L_{y(u)}\langle F\rangle = H) P(L_{y(u)}\langle F\rangle = H) \\ &< \sum_{H\colon \eta\text{-good}} 1 \cdot P(L_{y(u)}\langle F\rangle = H) + \sum_{H\colon \text{not } \eta\text{-good}} \eta P(L_{y(u)}\langle F\rangle = H) \\ &= 1 \cdot P(L_{y(u)}\langle F\rangle \text{ is } \eta\text{-good}) + \eta P(L_{y(u)}\langle F\rangle \text{ is not } \eta\text{-good}) \\ &= (1-\eta) P(L_{y(u)}\langle F\rangle \text{ is } \eta\text{-good}) + \eta. \end{aligned}$$

Combining these two inequalities, we get $(1-\eta)P(L_{y(u)}\langle F\rangle \text{ is } \eta\text{-good}) + \eta > \theta_p - o(1)$, from which the required bound follows. □

LEMMA 3.8. *Let δ, $0 < \delta < 1$ be given and suppose $\theta_p > 1-\delta$. Then the probability is at least $1 - 2^{-\Omega(2^{\alpha m})}$ that there exists some $v_0 \in \mathit{Left}(\mathcal{B}_m)$ such that $\mathit{Conf}_F(v_0)$ is δ-dense.*

Proof. Choose η and ϵ so that $1 - \delta < \eta(1-2\epsilon) < \eta < \theta_p$. Consider the following experiment in which we expose nodes (more precisely, expose their membership to the fault set F) in three stages. Let $t = \lfloor \epsilon 2^{\alpha m} \rfloor$ and let Y be an arbitrary subset of $\{0,1\}^{m-s}$ with $|Y| = t$. In the first stage we look at the left subbutterflies L_y, $y \in Y$ and expose faulty nodes in those subbutterflies. For each $y \in Y$, designate $v_y \in \mathit{Left}(L_y)$ arbitrarily and let A_y denote the event that $L_y\langle F\rangle$ is η-good for v_y. Events A_y are mutually independent and $P(A_y) = \zeta - o(1)$, $\zeta > 0$ for each $y \in Y$ by Lemma 3.7. Therefore, with probability at least $1 - 2^{-\Omega(t)} = 1 - 2^{-\Omega(2^{\alpha m})}$, at least one of the events A_y occurs.

Suppose now that at least one of the events A_y does occur. Thus, we have some node v_0 such that $L_{y(v_0)}\langle F\rangle$ is η-good for v_0. We now enter the second stage of the experiment and expose nodes in the remaining $2^{m-s} - t$ subbutterflies L_y, $y \notin Y$. Let $V = \bigcup_{y \notin Y} \mathit{Left}(L_y)$ and let $U = \{u \in V \mid E_{uv_0}\}$.

Having fixed this set U, we enter the third stage and now expose nodes on levels higher than s. By Lemma 3.6, for each $u \in U$, the probability that u is confluent with v_0 is at least $1 - 2^{-2^{\alpha m}}$ and therefore, the probability that every $u \in U$ is confluent with v_0 is at least $1 - 2^m 2^{-2^{\alpha m}} = 1 - 2^{-\Omega(2^{\alpha m})}$. Thus, with at least this probability, $U \subseteq \mathit{Conf}_F(v_0)$.

Now we return to the second stage and show that U is δ-dense with high probability. Fix $i \in [m]$ and $b \in \{0,1\}$ and let $V' = \{\langle 0, z\rangle \in V \mid \text{the } i\text{th bit of } z \text{ is } b\}$. Note $|V'| \geq 2^{m-1} - \epsilon 2^m = (1-2\epsilon)2^{m-1}$ and therefore, the expected number of nodes $u \in V'$ such that E_{uv_0} occurs is at least $\eta(1-2\epsilon)2^{m-1}$ because $L_{y(v_0)}$ is η-good for v_0. Noting that $\eta(1-2\epsilon) > 1-\delta$, we estimate the tail probability that the number of such nodes is less than $(1-\delta)2^{m-1}$ in the following way. Partition V' into 2^s subsets V'_x, $x \in \{0,1\}^s$, where $\langle 0, z\rangle$ is in V'_x if and only if x is the length s prefix of z. Thus, each subset contains at most one node from each of the subbutterflies L_y, $y \in Y$, and the size of each subset is at least $(1-\epsilon)2^{m-t-1} = \Omega(2^{\alpha m})$. Since the events E_{uv_0} are mutually independent for nodes u belonging to a single subset, we can apply the partition Chernoff bound (Proposition 2.3): the probability is $1 - 2^{-\Omega(2^{\alpha m})}$ that we have at least $(1-\delta)2^{m-1}$ nodes $u \in V'$ for which E_{uv_0} occurs. The probability that this is true for all choices of i and b, hence implying that U is δ-dense, is at least $1 - 2m2^{-\Omega(2^{\alpha m})} = 1 - 2^{-\Omega(2^{\alpha m})}$. □

We have shown that Φ_m succeeds on $\mathcal{B}_m\langle p\rangle$ with probability at least $1-2^{-2^{\Omega(m)}}$ which together with Lemma 3.2 establishes Theorem 3.1.

COROLLARY 3.9. *For any positive constant $\delta < 1/2$ and a positive constant p such that $\theta_p > 1-\delta$, the probability is $1-2^{-2^{\Omega(m)}}$ that $\mathcal{B}_m\langle p\rangle$ contains a δ-dense set U_j on each level $j \in [0,\dots,m]$ such that $\bigcup_j U_j$ is connected.*

Proof. Let F be randomly chosen from $\mathbf{randsub}(V(\mathcal{B}_m), p)$. For each $j \in [0,\dots,\lceil m/2\rceil]$, apply the theorem to each subbutterfly starting at level j and ending at level m. Because there are only $O(2^m)$ subbutterflies to consider and each subbutterfly has at least $m/2$ stages, each of those subbutterflies will yield a δ-dense subset of its left end, which is connected in $\mathcal{B}_m\langle F\rangle$ with probability $1-2^{-2^{\Omega(m)}}$. Collect these δ-dense subsets from all subbutterflies starting at level j and call their union U_j. Then U_j is δ-dense by definition. We claim that $U = \bigcup_j U_j$ is connected in $\mathcal{B}_m\langle F\rangle$. To see this, focus on a particular subbutterfly starting at level j, $j \le \lceil m/2\rceil - 1$ and on one of its two subbutterflies starting at level $j+1$. The δ-dense sets of these subbutterflies we have obtained share at least one row, since $\delta < 1/2$ by the assumption; hence, these sets are connected through a straight edge. Applying this argument to every interface of two subbutterflies starting at consecutive levels, we conclude that U is connected. Applying the argument symmetrically to the right half of $\mathcal{B}_m\langle F\rangle$, we obtain a set U' that is connected in $\mathcal{B}_m\langle F\rangle$ and includes a δ-dense set on each level j, $\lfloor m/2\rfloor \le j \le m$. Now, U and U' share at least one node and thus are connected. □

4. $(O(m^{\log 6}), m^{O(1)})$-routing. In this section, we prove the following theorem which improves the congestion of routing from $O(2^m)$ of the previous section to $m^{O(1)}$, at the expense of the path length of routing. Due to the inductive structure of the proof, we need to consider end-to-end routings instead of routings among nodes at one end of the butterfly.

THEOREM 4.1. *Let δ and p be positive constants satisfying $\delta < 1/2$ and $\theta_p > 1-\delta$. Then, with probability $1-2^{-2^{\Omega(m)}}$, $\mathcal{B}_m\langle p\rangle$ contains a δ-dense set U of nodes on level 0 and a δ-dense set V of nodes on level m such that (U,V) is $(O(m^{\log 6}), m^{O(1)})$-routable.*

Fix $\delta < 1/2$. In the following, we define a mapping Ψ_m that takes a faulty butterfly $\mathcal{B}_m\langle F\rangle$ and is supposed to give a pair (U,V) such that U and V are δ-dense subsets of $\mathit{Left}(\mathcal{B}_m)$ and $\mathit{Right}(\mathcal{B}_m)$, respectively, and such that (U,V) is $(O(m^{\log 6}), m^{O(1)})$-routable in $\mathcal{B}_m\langle F\rangle$. We will later show that Ψ_m succeeds in giving such a pair with high probability when applied to $\mathcal{B}_m\langle p\rangle$, where p is sufficiently small.

The definition of Ψ_m is by induction on m. The induction is based on the following structure of the butterfly.

First consider the $2m$-stage butterfly $\mathcal{B}_{2m}$. Split this butterfly at level m, let L_y, $y \in \{0,1\}^m$ be the left m-stage subbutterflies, and let R_x, $x \in \{0,1\}^m$ be the right m-stage subbutterflies. As before, the node set of L_y is $\{\langle i, xy\rangle \mid x \in \{0,1\}^m, i \in [0,\dots,m]\}$ and the node set of R_x is $\{\langle i, xy\rangle \mid y \in \{0,1\}^m, i \in [m,\dots,2m]\}$. Observe as before that each pair (L_y, R_x) intersects at exactly one node $\langle m, xy\rangle$. To depict this global structure of $\mathcal{B}_{2m}$, we define a complete bipartite graph $\mathcal{S}_{2m}$ on node sets $\{l_y \mid y \in \{0,1\}^m\}$ and $\{r_x \mid x \in \{0,1\}^m\}$. The intention here is to associate left subbutterfly L_y with node l_y, right subbutterfly R_x with node r_x, and the intersection node of L_y and R_x with edge (l_y, r_x) of $\mathcal{S}_{2m}$. We call $\mathcal{S}_{2m}$ the *skeleton graph* of $\mathcal{B}_{2m}$.

We have similar definitions for the odd-stage butterfly $\mathcal{B}_{2m+1}$. The left subbutterfly L_y, $y \in \{0,1\}^{m+1}$ is an m-stage subbutterfly consisting of the nodes $\{\langle i, x'y\rangle \mid x' \in$

$\{0,1\}^m, i \in [0,\ldots,m]\}$ and the right subbutterfly R_x, $x \in \{0,1\}^{m+1}$ is an m-stage subbutterfly consisting of the nodes $\{\langle i, xy'\rangle \mid y' \in \{0,1\}^m, i \in [m+1,\ldots,2m]\}$. Observe that each pair (L_y, R_x) is connected by exactly one edge, $(\langle m, x'y\rangle, \langle m+1, xy'\rangle)$, where x' is the length m prefix of x and y' is the length m suffix of y. The skeleton graph $\mathcal{S}_{2m+1}$ is defined similarly, with edge (l_y, r_x) of $\mathcal{S}_{2m+1}$ now corresponding to the edge of $\mathcal{B}_{2m+1}$ that connects L_y and R_x.

As we remove faulty nodes from the butterfly $\mathcal{B}_{2m}$ or $\mathcal{B}_{2m+1}$, we will remove nodes and edges of its skeleton graph, in a manner to be made precise later. Roughly speaking, if the shared node of L_y and R_x of $\mathcal{B}_{2m}$ is removed or disconnected from the major connected component of L_y or R_x, then we remove edge (l_y, r_x) of the skeleton graph. If the resulting subgraph of the skeleton graph is dense, we can expect that the faulty butterfly is still capable of efficient routing. This motivates the following definition of a measure of the density of a bipartite graph.

Let u and v be nodes on the same side of a bipartite graph. The *codegree* of u and v is the number of common neighbors of u and v. The *minimum codegree* of a bipartite graph is the minimum of codegrees of all pairs (u, v) where u and v are nodes on the same side of the bipartite graph.

Now we are ready to define the mapping Ψ_m. The definition depends on positive constants δ, g, p, α, m_0, and β, chosen as follows. We have already fixed $\delta < 1/2$. Fix $g \le 1/2$ and, according to Corollary 3.9, choose positive constants p, $\alpha < 1/2$, and m_0' so that $\mathcal{B}_m\langle p\rangle$ contains a connected set that includes a $\delta(1-g)$-dense set at each level with probability at least $1 - 2^{-2^{\alpha m}}$ for every $m \ge m_0'$. Choose $\beta < (1-2\delta)^2$. Let $m_0 \ge m_0'$ be a constant that satisfies the following conditions. Let $\mu_m = 2^{-2^{\alpha m}} 2^{m-1}$ and $T_m = \delta g (2g^{1/m_0})^m/2$. We require that for every $m \ge m_0$,

M1 $T_m \ge 2^{2\alpha m} + \log m + 3$,
M2 $e\mu_m/T_m < 1/2$,
M3 $2^{-2^{\alpha m}} \le \delta g^m$, and
M4 $e^{-c_\epsilon (1-2\delta)^2 2^m} \le 2^{-(2^{\alpha m}+2m+2)}$, where $\epsilon = (1-2\delta)^2/\beta - 1$ and c_ϵ is the constant in Proposition 2.1.

The meaning of these conditions will become clear when they are used later. We note here that all of these conditions can be satisfied by choosing m_0 sufficiently large.

Define $\delta_m = \delta(1 - g^{\frac{m}{m_0}})$ for every $m \ge m_0$. Note that δ_m is increasing in m and less than δ.

As the base case, we first define Ψ_m for $m_0 \le m < 2m_0$. Consider $\mathcal{B}_m$, $m_0 \le m < 2m_0$, and fix a fault set $F \subseteq V(\mathcal{B}_m)$. Suppose $\mathcal{B}_m\langle F\rangle$ contains a connected component C such that $C \cap \mathit{Left}(\mathcal{B}_m)$ and $C \cap \mathit{Right}(\mathcal{B}_m)$ are both δ_m-dense. Then, set $\Psi_m(\mathcal{B}_m\langle F\rangle) = (C \cap \mathit{Left}(\mathcal{B}_m), C \cap \mathit{Right}(\mathcal{B}_m))$, choosing one such C with an arbitrary criterion. Otherwise set $\Psi_m(\mathcal{B}_m\langle F\rangle) = (\emptyset, \emptyset)$, indicating the failure of the mapping.

Suppose Ψ_m is already defined. Now we define Ψ_{2m} and Ψ_{2m+1}. We first deal with Ψ_{2m}. Fix a fault set $F \subseteq V(\mathcal{B}_{2m})$. We apply Ψ_m to each left subbutterfly $L_y\langle F\rangle$ and each right subbutterfly $R_x\langle F\rangle$ of $\mathcal{B}_{2m}\langle F\rangle$ through the standard isomorphisms. More precisely, for each $y \in \{0,1\}^m$, let ψ_y denote the standard isomorphism mapping L_y to $\mathcal{B}_m$ and set $(U_y, V_y) = \psi_y^{-1}(\Psi_m(\psi_y(L_y\langle F\rangle)))$. Similarly for each $x \in \{0,1\}^m$, let ψ_x' denote the standard isomorphism mapping R_x to $\mathcal{B}_m$ and set $(W_x, Z_x) = \psi_x'^{-1}(\Psi_m(\psi_x'(R_x\langle F\rangle)))$. The *skeleton graph* $\mathcal{S}_{2m}(F)$ *of the faulty butterfly* $\mathcal{B}_{2m}\langle F\rangle$ is a subgraph of $\mathcal{S}_{2m}$, where (1) node l_y is included if and only if $V_y \ne \emptyset$; (2) node r_x is included if and only if $W_x \ne \emptyset$; and (3) edge (l_y, r_x) is included if and only if $V_y \cap W_x \ne \emptyset$. We set

$$\Psi_{2m}(\mathcal{B}_{2m}\langle F\rangle) = \left(\bigcup_{y\in\{0,1\}^m} U_y, \bigcup_{x\in\{0,1\}^m} Z_x\right),$$

if the minimum codegree of $\mathcal{S}_{2m}(F)$ is at least $\beta 2^m$ and, moreover, $\bigcup_y U_y$ and $\bigcup_x Z_x$ are both δ_{2m}-dense; otherwise set

$$\Psi_{2m}(\mathcal{B}_{2m}\langle F\rangle) = (\emptyset, \emptyset).$$

Ψ_{2m+1} is defined similarly. We apply Ψ_m to subbutterflies $L_y\langle F\rangle$ and $R_x\langle F\rangle$ of $\mathcal{B}_{2m+1}\langle F\rangle$ to obtain pairs (U_y, V_y) and (W_x, Z_x). The skeleton graph $\mathcal{S}_{2m+1}(F)$ is also defined similarly as above, with item (3) replaced by: (3′) edge (l_y, r_x) is included if and only if there is some $v \in V_y$ and some $w \in W_x$ such that (v, w) is an edge of $\mathcal{B}_{2m+1}$. Then, set $\Psi_{2m+1}(\mathcal{B}_{2m+1}\langle F\rangle) = (\bigcup_y U_y, \bigcup_x Z_x)$ if the minimum codegree of $\mathcal{S}_{2m+1}(F)$ is at least $\beta 2^{m+1}$ and, moreover, $\bigcup_y U_y$ and $\bigcup_x Z_x$ are both δ_{2m+1}-dense; otherwise set $\Psi_{2m+1}(\mathcal{B}_{2m+1}\langle F\rangle) = (\emptyset, \emptyset)$.

It is easy to see that we can compute $\Psi_m(\mathcal{B}_m\langle F\rangle)$ for given $\mathcal{B}_m\langle F\rangle$ in time polynomial in the number of nodes of $\mathcal{B}_m$.

We say that Ψ_m *succeeds on* $\mathcal{B}_m\langle F\rangle$ if $\Psi_m(\mathcal{B}_m\langle F\rangle) \neq (\emptyset, \emptyset)$; otherwise it *fails*. By definition, Ψ_m gives δ-dense sets when it succeeds. To show that it also gives an $(O(m^{\log 6}), m^{O(1)})$-routable pair, we first need the following lemma, which is intended to be applied to the skeleton bipartite graph of a faulty butterfly.

LEMMA 4.2. *Let G be a bipartite graph on node sets A and B such that $|A|, |B| \leq n$ and the minimum codegree of G is at least βn, where β is some positive constant. Let Q be an arbitrary multiset of pairs from $A \times B$ in which each node appears at most n times. Then we can route all pairs of Q in G with a multiset of paths of length 3 so that each edge is used at most Γ_β times, where Γ_β is some constant depending only on β.*

Proof. Assume, without loss of generality, that $2/\beta$ is an integer (otherwise reason with the largest $\beta' < \beta$ for which this is true) and set $\Gamma_\beta = \frac{4}{\beta}(\frac{4}{\beta} + 1)$. We prove a stronger claim by induction on the size of Q: the multiset of paths described above can be chosen to have an additional property that each node appears as an internal node of at most $2n/\beta$ paths from the multiset.

The base case $Q = \emptyset$ is trivial. Suppose Q is nonempty and the claim holds for all Q' with $|Q'| = |Q| - 1$. Removing one pair (a, b) of Q, we can route the remaining pairs of Q in G with paths Π, as in the claim, by the induction hypothesis. We show that we can add a path for (a, b) to Π without violating the constraints in the claim. We say that an edge of G is *saturated* (with respect to Π) if the edge appears Γ_β times in Π. Similarly, a node of G is *saturated* if it appears as an internal node in $2n/\beta$ paths of Π. Because each node appears at most n times in Q, Q contains at most n^2 pairs. Therefore, the number of paths in Π which equals the number of pairs in $Q \setminus \{(a, b)\}$ is strictly less than n^2. Because each path has exactly one internal node in A, these paths in Π can saturate strictly fewer than $\beta n/2$ nodes of A; the same holds for B. Next consider the set of edges incident to an arbitrary node c. Because at most n paths of Π end at c and at most $2n/\beta$ paths go through c, these edges are used at most $(\frac{4}{\beta} + 1)n$ times in total. Therefore, at most $(\frac{4}{\beta} + 1)n/\Gamma_\beta = \beta n/4$ edges incident to c are saturated.

Now we can choose the path for (a, b). Since a has at least βn edges, of which at least $\beta n - \beta n/4 > \beta n/2$ are unsaturated, there is at least one unsaturated neighbor, say b', such that the edge (a, b') is unsaturated. We take this b' as the first intermediate

node in the path. Nodes b and b' have at least βn common neighbors, of which at least $\beta n - \beta n/4 - \beta n/4 = \beta n/2$ are reachable from both b' and b through unsaturated edges. One of these $\beta n/2$ neighbors must be unsaturated, completing a path for (a, b) free of saturated edges or nodes. Adding this path to Π preserves the conditions in the claim. □

LEMMA 4.3. *If Ψ_m succeeds on $\mathcal{B}_m\langle F\rangle$ and $\Psi_m(\mathcal{B}_m\langle F\rangle) = (U, V)$, then (U, V) is $(c_1 m^{\log 6}, c_2 m^{\log \Gamma_\beta})$-routable in $\mathcal{B}_m\langle F\rangle$ where c_1 and c_2 are some constants independent of m, β is the constant fixed in the definition of Ψ_m, and Γ_β is the constant given in Lemma 4.2.*

Proof. The proof is by induction on m. Since m_0 is a constant, we can choose c_1 and c_2 so that the statement holds for all Ψ_m in the base cases $m_0 \le m < 2m_0$. Suppose the statement holds for Ψ_m. We first show that it also holds for Ψ_{2m}. Fix a faulty butterfly $\mathcal{B}_{2m}\langle F\rangle$, let $\Psi_{2m}(\mathcal{B}_{2m}\langle F\rangle) = (U, V)$, and assume U and V are nonempty. Recall the definition of $\Psi_{2m}(\mathcal{B}_{2m}\langle F\rangle)$ and notation used there. In particular, (U_y, V_y), $y \in \{0,1\}^m$ is the pair produced by Ψ_m applied to the left subbutterfly $L_y\langle F\rangle$ of $\mathcal{B}_{2m}\langle F\rangle$, and (W_x, Z_x), $x \in \{0,1\}^m$ is the pair for the right subbutterfly $R_x\langle F\rangle$. Let $A = \{l_y \mid V_y \neq \emptyset\}$ and $B = \{r_x \mid W_x \neq \emptyset\}$. Recall that A and B are the node sets of the skeleton graph $\mathcal{S}_{2m}(F)$.

Now, let $Q \subseteq U \times V$ be an arbitrary one-to-one routing request. We define a many-to-many routing request $\hat{Q}$ on (A, B) induced by Q as follows. For each node $u \in U$, let $l(u)$ denote $l_y \in A$ such that u belongs to L_y and, for each node $v \in V$, let $r(v)$ denote $r_x \in B$ such that v belongs to R_x. Now, we construct multiset $\hat{Q}$ by putting an occurrence of $(l(u), r(v))$ for each pair (u, v) of Q. Then, each l_y or r_x appears at most 2^m times in $\hat{Q}$. On the other hand, by the definition of Ψ_m, the minimum codegree of $\mathcal{S}_{2m}(F)$ is at least $\beta 2^m$. Therefore, we can apply Lemma 4.2 to $\hat{Q}$ and $\mathcal{S}_{2m}(F)$, with $n = 2^m$, obtaining a realization of $\hat{Q}$ with paths of length exactly 3 and with congestion at most Γ_β.

We now translate this realization of $\hat{Q}$ on $\mathcal{S}_{2m}(F)$ back to the realization of Q on $\mathcal{B}_{2m}\langle F\rangle$. Suppose $(l_y, r_{x'}, l_{y'}, r_x)$ is the path in $\mathcal{S}_{2m}(F)$ assigned to pair $(l(u), r(v)) \in \hat{Q}$. The path from u to v in our realization of Q consists of four parts: a path in L_y from u to the intersection node of L_y and $R_{x'}$; a path in $R_{x'}$ from this node to the intersection node of $R_{x'}$ and $L_{y'}$ via some node in $Z_{x'}$; a path from this intersection node to the intersection node of $L_{y'}$ and R_x via some node in $U_{y'}$; and finally a path in R_x from this intersection node to v. (See Figure 4.) When we provide such paths for all pairs in Q, the number of times the intersection node of a particular pair L_y and R_x appears in the above path construction is exactly the congestion of the edge (l_y, r_x) in our realization of $\hat{Q}$ in $\mathcal{S}_{2m}(F)$, which is at most Γ_β. Thus, within each subbutterfly L_y (or R_x), the total routing request can be decomposed into at most Γ_β one-to-one routing requests for the pair (U_y, V_y) (or (W_x, Z_x), respectively). Because all pairs (U_y, V_y) and (W_x, Z_x) involved are $(c_1 m^{\log 6}, c_2 m^{\log \Gamma_\beta})$-routable by the induction hypothesis, each subbutterfly can realize all the routing requirements with path length at most $c_1 m^{\log 6}$ and congestion at most $\Gamma_\beta c_2 m^{\log \Gamma_\beta} = c_2 (2m)^{\log \Gamma_\beta}$. Since the path assigned to a pair $(l(u), r(v))$ is a concatenation of six end-to-end routing paths in the subbutterflies, its length is at most $6 \cdot c_1 m^{\log 6} = c_1 (2m)^{\log 6}$. This completes the proof that Ψ_{2m} produces a $(c_1 (2m)^{\log 6}, c_2 (2m)^{\log \Gamma_\beta})$-routable pair when it succeeds.

The proof for Ψ_{2m+1} is similar and therefore omitted. We note that we need a variation of Lemma 4.2 which assumes each node to appear in the routing request at most $n/2$ times and gives the congestion bound of $\Gamma_\beta/2$. □

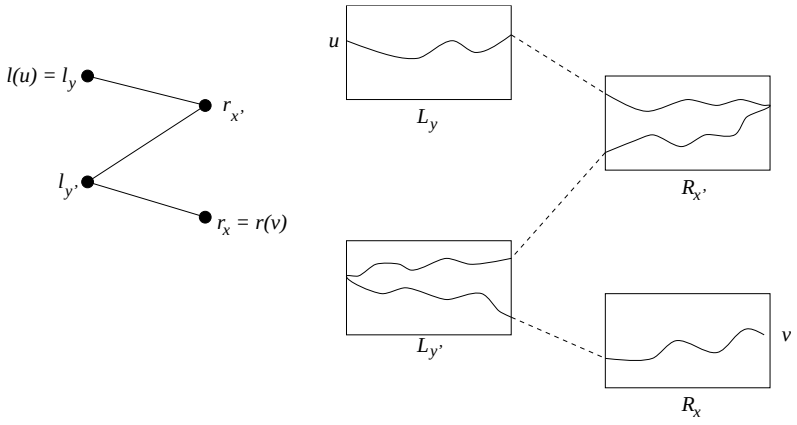


FIG. 4. *Bipartite routing to butterfly routing.*

It remains to show that the probability of Ψ_m failing is small.

LEMMA 4.4. *Let p and α be constants as fixed in the definition of Ψ_m. Then, the probability that $\Psi_m(\mathcal{B}_m\langle p\rangle) = (\emptyset, \emptyset)$ is at most $2^{-2^{\alpha m}}$.*

Proof. The proof is by induction on m. The base cases, $m_0 \le m < 2m_0$, are immediate from the definition of Ψ_m.

For the induction step, we assume that the claim holds for Ψ_m, $m \ge m_0$ and show that it also holds for Ψ_{2m} and Ψ_{2m+1}. We first consider Ψ_{2m}. Let $F \in$ **randsub**$(V(\mathcal{B}_{2m}), p)$, let $\Psi_{2m}(\mathcal{B}_{2m}\langle F\rangle) = (U, V)$, and recall the notation in the definition of Ψ_{2m}. More specifically, Ψ_m applied to the left subbutterfly $L_y\langle F\rangle$ yields (U_y, V_y) and, similarly, Ψ_m applied to the right subbutterfly $R_x\langle F\rangle$ yields (W_x, Z_x). There are two ways in which $\Psi_{2m}(\mathcal{B}_{2m}\langle F\rangle)$ fails: (1) Ψ_m applied to left and right subbutterflies fails on too many subbutterflies and thus $\bigcup_y U_y$ or $\bigcup_x V_x$ fails to be δ_{2m}-dense; (2) the minimum codegree of the skeleton graph of $\mathcal{B}_{2m}\langle F\rangle$ is less than $\beta 2^m$.

We bound the probability of each of these cases in turn.

We first bound the probability that $\bigcup_y U_y$ is not δ_{2m}-dense. Recall that δ_m is defined to be $\delta(1 - g^{\frac{m}{m_0}})$ for every $m \ge m_0$. Now define $\epsilon_m = \delta g^{(\frac{m}{m_0}+1)}$ so that $\delta_m + \epsilon_m \le \delta_{2m}$. Let Y_i^b, $i \in [m]$, $b \in \{0,1\}$, denote the set of bit strings $y \in \{0,1\}^m$ such that the ith bit of y is b and $U_y = \emptyset$.

CLAIM 4.5. *If $\bigcup_y U_y$ is not δ_{2m}-dense, then $|Y_i^b| \ge \epsilon_m 2^{m-1}$ must hold for some i and b.*

Proof. Suppose $\bigcup_y U_y$ is not δ_{2m}-dense. Then, for some $h \in [2m]$ and $b' \in \{0,1\}$, the number of nodes of $\bigcup_y U_y$ whose row indices have b' as their hth bit must be less than $(1-\delta_{2m})2^{2m-1}$. Fix h and b' such that this holds and call this set of nodes U. We have two cases.

Case $h \le m$. Because each nonempty U_y, $y \in \{0,1\}^m$ is δ_m-dense, each such U_y contributes at least $(1-\delta_m)2^{m-1}$ nodes to U. Therefore, U_y must be empty for at least $\epsilon_m 2^m$ values of y because otherwise, $|U| \ge (1-\epsilon_m)2^m(1-\delta_m)2^{m-1} \ge (1-\delta_{2m})2^{2m-1}$, contradicting our assumption. Therefore, for any $i \in [m]$, we have $|Y_i^0 \cup Y_i^1| \ge \epsilon_m 2^m$. Choose any i and choose an appropriate b so that $|Y_i^b| \ge \epsilon_m 2^{m-1}$.

Case $h > m$. Set $i = h - m$ and $b = b'$. Note that each U_y is entirely included in U if the ith bit of y is b and is entirely excluded from U otherwise. Because each

nonempty U_y has at least $(1-\delta_m)2^m$ nodes, U_y must be empty for at least $\epsilon_m 2^{m-1}$ values of y such that the ith bit of y is b. □

Fix i and b. We bound the probability of event $|Y_i^b| \geq \epsilon_m 2^{m-1}$. Note the probability of $U_y = \emptyset$ for a fixed y is at most $2^{-2^{\alpha m}}$ by the induction hypothesis and, since this event is independent among $y \in Y_i^b$, we can use the Chernoff bound.

Applying Proposition 2.2, we have

$$P(|Y_i^b| \geq T_m) < \left(\frac{e\mu_m}{T_m}\right)^{T_m},$$

where $\mu_m = 2^{-2^{\alpha m}} 2^{m-1}$ and $T_m = \epsilon_m 2^{m-1}$. By condition M1 on the choice of m_0, we have $T_m \geq 2^{2\alpha m} + \log m + 3$. Moreover, we have $e\mu_m/T_m \leq 1/2$ by condition M2. Combining these inequalities, we get

$$P(|Y_i^b| \geq \epsilon_m 2^{m-1}) < \frac{1}{8m} 2^{-2^{2\alpha m}},$$

and because we have only $2m$ pairs (i,b) to consider,

$$P(\textstyle\bigcup_y U_y \text{ is not } \delta_{2m}\text{-dense}) < 2^{-2^{2\alpha m}-2}.$$

The bound for the probability that $\bigcup_x Z_x$ is not δ_{2m}-dense is the same.

We now bound the probability that the minimum codegree of $\mathcal{B}_{2m}\langle F\rangle$ is less than $\beta 2^m$. Let $v_{xy} = \langle m, xy\rangle$ denote the intersection node of L_y and R_x for each pair $x, y \in \{0,1\}^m$. We first claim that $P(v_{xy} \in W_x) \geq 1-\delta$. To show this, first note that Ψ_m gives δ_m-dense sets with probability at least $1 - 2^{-2^{\alpha m}}$ when applied to $R_x\langle F\rangle$. Each node of $\mathit{Left}(R_x)$ is equally likely to be in W_x due to the symmetry of Ψ_m with respect to automorphisms of $\mathcal{B}_m$. Thus, conditional on the event that W_x is δ_m-dense, which implies that $|W_x|/|\mathit{Left}(R_x)| \geq 1-\delta_m$, the probability that $v_{xy} \in W_x$ is at least $1-\delta_m$. It follows that the probability that $v_{xy} \in W_x$ is at least $1 - 2^{-2^{\alpha m}} - \delta_m$. By condition M3 on the choice of m, we have $2^{-2^{\alpha m}} + \delta_m \leq \delta(g^m + 1 - g^{\frac{m}{m_0}}) \leq \delta$, establishing the claim.

Fix distinct bit strings $y, y' \in \{0,1\}^m$. We bound the probability that nodes l_y and $l_{y'}$ are both in the skeleton graph $\mathcal{S}_{2m}(F)$ and the codegree of l_y and $l_{y'}$ is smaller than $\beta 2^m$. This probability is at most the conditional probability that the codegree of l_y and $l_{y'}$ is smaller than $\beta 2^m$ under the condition that l_y and $l_{y'}$ are nodes of $\mathcal{S}_{2m}(F)$. To bound this conditional probability, consider an experiment in which we expose the membership of nodes to F in two stages: in the first stage we expose the nodes in the left subbutterflies L_y and $L_{y'}$ and in the second stage we expose the remaining nodes. The first stage determines whether or not l_y and $l_{y'}$ are nodes of $\mathcal{S}_{2m}(F)$. Suppose that l_y and $l_{y'}$ are nodes of $\mathcal{S}_{2m}(F)$. Let $X_{yy'} = \{x \mid v_{xy} \in V_y, v_{xy'} \in V_{y'}\}$. Because both V_y and $V_{y'}$ are nonempty, and hence δ_m-dense, by the definitions of the skeleton graph and the mapping Ψ_m, we have $|V_y|, |V_{y'}| \geq (1-\delta)2^m$ and therefore $|X_{yy'}| \geq (1-2\delta)2^m$. Now consider R_x where $x \in X_{yy'}$. The only nodes of R_x that are exposed in the first stage of the experiment are v_{xy} and $v_{xy'}$. The condition $x \in X_{yy'}$ implies that these two nodes are revealed to be *not* in F. Thus, since the event that a fixed node is in W_x is increasing, the inequalities $P(v_{xy} \in W_x) \geq 1-\delta$ and $P(v_{xy'} \in W_x) \geq 1-\delta$ that we observed earlier are valid even when conditioned on the result of the first stage of the experiment. Therefore, we have

$$P(v_{xy}, v_{xy'} \in W_x) \geq 1 - 2\delta$$

in the second stage. Thus, the expected number of values of x such that $x \in X_{yy'}$ and $v_{xy}, v_{xy'} \in W_x$ is at least $(1-2\delta)^2 2^m$. Since the event in the second stage that both v_{xy} and $v_{xy'}$ are in W_x is mutually independent for different values of $x \in X_{yy'}$ and since $(1-2\delta)^2 > \beta$ by our choice of β, we can apply the Chernoff bound: the probability is at most $2^{-\Omega(2^m)}$ that the number of values of $x \in X_{yy'}$ such that $v_{xy}, v_{xy'} \in W_x$, or equivalently the codegree of l_y and $l_{y'}$, is less than $\beta 2^m$. By condition M4 on the choice of m_0, this probability bound is at most $2^{-(2^{\alpha m}+2m+2)}$.

Summing up this probability for all pairs of distinct $y, y' \in \{0,1\}^m$ and also considering the codegree of the pairs $r_x, r_{x'}$, the overall probability that the codegree condition is violated anywhere is at most $2^{-2^{\alpha m}-1}$.

Summing up the above two cases, we conclude that the probability that Ψ_{2m} fails is at most $2^{-2^{\alpha m}}$.

The analysis for Ψ_{2m+1} is similar and is omitted. □

Finally, Theorem 4.1 follows immediately from Lemmas 4.3 and 4.4.

5. Embedding. In this section we prove our main embedding results using the routing results in the previous sections.

The overall structure of our embedding is as follows. We look at the n-stage butterfly $\mathcal{B}_n$ as a collection of overlapping m-stage subbutterflies, where $m = \Theta(\log n)$. We partition each level of $\mathcal{B}_n$ into *blocks* based on these subbutterflies. A set of nodes is a *left-block* if it is *Left*(B) for some m-stage subbutterfly B; it is a *right-block* if it is *Right*(B) for some m-stage butterfly B. In our embedding, a node on level $i \in [0, \ldots, n-m]$ is mapped to a node within the left-block it belongs to; an edge in level $i \in [1, \ldots, n-m]$ joining two left-blocks *Left*(B) and *Left*(B') is mapped to a path that goes through B and B'. We call this part of the embedding the *left-embedding*. The mapping of nodes and edges in the last m stages are similar, with right-blocks replacing the role of left-blocks in the left-embedding. We call this part of the embedding the *right-embedding*. However, in the above scheme, the left- and right-embeddings give two possibly inconsistent mappings of nodes at level $n-m$. We resolve this problem by providing a path between the two images of each node on level $n-m$.

Let us formalize the above embedding strategy. A *range assignment* ρ for $\mathcal{B}_n$ is a pair of mappings (ρ_l, ρ_r), where ρ_l assigns to each m-stage subbutterfly B a subset of *Left*(B), and ρ_r assigns to each m-stage subbutterfly B ending at level $i \in [n-m, \ldots, n]$ a subset of *Right*(B). The intention of a range assignment ρ is to specify the range of the node mapping in our embedding; we map a node in each *Left*(B) to a node in $\rho_l(B)$ and map a node in each *Right*(B), where B is ending at level $i \in [n-m, \ldots, n]$, to a node in $\rho_r(B)$.

We say that a range assignment ρ is $(\lambda, \gamma, \delta, \beta)$-*good* for $\mathcal{B}_n\langle F\rangle$ if it satisfies the following three conditions C1, C2, and C3; we first describe the first two conditions.

- **C1** For every m-stage subbutterfly B of $\mathcal{B}_n$, $\rho_l(B)$ is disjoint from F, δ-dense, and (λ, γ)-routable in $B\langle F\rangle$.
- **C2** For every m-stage subbutterfly B of $\mathcal{B}_n$ that ends at level $i \in [n-m, \ldots, n]$, $\rho_r(B)$ is disjoint from F, δ-dense, and (λ, γ)-routable in $B\langle F\rangle$.

Condition C1 will ensure the existence of an efficient left-embedding and condition C2 an efficient right-embedding. To describe the third condition which would allow us to efficiently resolve the inconsistency of the left- and right-embeddings, we need some definitions. Let D be a $2m$-stage subbutterfly that ends at level n, let L_y, $y \in \{0,1\}^m$ be the left m-stage subbutterflies of D, and let R_x, $x \in \{0,1\}^m$ be the right m-stage subbutterflies of D. Define a bipartite graph $\mathcal{S}_D(\rho)$ to be a subgraph of the skeleton

graph $\mathcal{S}_{2m}$ in which all nodes are included and edge (l_y, r_x) is included if and only if $\rho_r(L_y) \cap \rho_l(R_x) \neq \emptyset$.

C3 For each $2m$-stage subbutterfly D that ends at level n, the bipartite graph $\mathcal{S}_D(\rho)$ has co-degree at least $\beta 2^m$.

LEMMA 5.1. *Let ρ be a range assignment that is $(\lambda, \gamma, \delta, \beta)$-good for $\mathcal{B}_n\langle F\rangle$. Let D be a $2m$-stage subbutterfly that ends at level n, let L_y, $y \in \{0,1\}^m$ be the left m-stage subbutterflies of D, and let R_x, $x \in \{0,1\}^m$ be the right m-stage subbutterflies of D. Then the set of nodes $\bigcup_y \rho_r(L_y) \cup \bigcup_x \rho_l(R_x)$ on level $n-m$ is $(8\lambda, (\Gamma_\beta+1)\gamma)$-routable in $D\langle F\rangle$, where Γ_β is the constant that depends on β as given in Lemma 4.2.*

Proof. The proof uses Lemma 4.2 and is similar to the proof of Lemma 4.3. The details are omitted. □

LEMMA 5.2. *If there exists a $(\lambda, \gamma, \delta, \beta)$-good range assignment for $\mathcal{B}_n\langle F\rangle$, where $\delta < 1/2$ and β are positive constants, then $\mathcal{B}_n$ can be embedded in $\mathcal{B}_n\langle F\rangle$ with $O(1)$ load, $O(\lambda)$ dilation, and $O(m\gamma)$ congestion.*

Proof. Let ρ be a $(\lambda, \gamma, \delta, \beta)$-good range assignment for $\mathcal{B}_n\langle F\rangle$. We first provide a left-embedding. Since $\rho_l(B)$ is δ-dense ($\delta < 1/2$) for each m-stage subbutterfly B, for each B we can map nodes of the left-block *Left*(B) to nodes of $\rho_l(B)$ with load at most 2; choose and fix such a mapping for each B. To provide the mapping of edges, consider an m-stage subbutterfly B that starts with level i, $i < n-m$. The left-block *Left*(B) is adjacent to two left-blocks on level $(i+1)$, one of which we call *Left*(B'). There are 2^m edges between *Left*(B) and *Left*(B') of which 2^{m-1} are straight edges and the rest are cross edges. We call a straight edge a *bridge* between these two blocks if it spans $\rho_l(B)$ and $\rho_l(B')$. Because $\rho_l(B)$ and $\rho_l(B')$ are both δ-dense, there are at least $(1-2\delta)2^{m-1}$ bridges between *Left*(B) and *Left*(B'). We map each edge between *Left*(B) and *Left*(B') to a path consisting of three parts: a path in B, a bridge, and a path in B'. To choose such a path for each edge, we first choose a bridge (x, y) for each edge (u, v) so that each bridge is chosen for at most $2/(1-2\delta)$ edges. Let $u' \in \rho_l(B)$ and $v' \in \rho_l(B')$, respectively, denote the nodes to which u and v are mapped. Then we connect u' with x by a path of $B\langle F\rangle$ and connect y with v' by a path of $B'\langle F\rangle$. The routing requirements in $B\langle F\rangle$ generated by all edges between *Left*(B) and *Left*(B') can be decomposed into at most $2/(1-2\delta)$ one-to-one routing requests on $\rho_l(B)$, because each node of $\rho_l(B)$ appears at most that many times as the endpoint x of a bridge. Therefore, they can be realized by a set of paths with length at most $O(\lambda)$ and congestion at most $O(\gamma)$. The routing in $B'\langle F\rangle$ is done similarly. To provide paths for every adjacent pairs of blocks, we have to multiply the congestion by $4m$, because each block is adjacent to four blocks (to the right and to the left) and each edge belongs to m overlapping subbutterflies. We also have to add the congestion of an edge as a bridge, but this is at most a constant because an edge can be a bridge of at most one pair of blocks. Therefore, the overall congestion for the left-embedding is $O(m\gamma)$. The dilation is clearly $O(\lambda)$.

The right-embedding is provided similarly using condition C2 on the range assignment. Let ψ_1 denote the left-embedding and ψ_2 denote the right-embedding. Our final task is to provide a path between $\psi_1(v)$ and $\psi_2(v)$ for each node v on level $n-m$. Let D_v denote the $2m$-stage subbutterfly of $\mathcal{B}_n$ ending at level n and containing node v. Then, D_v also contains both $\psi_1(v)$ and $\psi_2(v)$. Now, for each such subbutterfly D, consider the routing request $Q = \{(\psi_1(v), \psi_2(v)) \mid D_v = D\}$. Since the loads of ψ_1 and ψ_2 are both 2, we can decompose Q into at most two one-to-one routing requests. Therefore, by Lemma 5.1 Q can be realized by paths of length $O(\lambda)$ and congestion $O(\gamma)$. Our final embedding maps a node v to $\psi_1(v)$ if v is on levels 0 up to $n-m$

and to $\psi_2(v)$ if v is on levels $n-m+1$ up to n; it maps an edge (u,v), where u is on level $n-m$ and v is on level $n-m+1$, to a path consisting of the path connecting $\psi_1(u)$ and $\psi_2(u)$ as given above, followed by the path $\psi_2(u,v)$. □

Now we are ready to prove our main theorem.

THEOREM 5.3. *For $p < 1-\sqrt{2/3}$, $\mathcal{B}_n\langle p\rangle$ admits a level-preserving self-embedding with $O(1)$ load, $O(\log^{\log 6} n)$ dilation, and $\log^{O(1)} n$ congestion, with probability at least $1-2^{-n}$.*

Proof. In view of Lemma 5.2, it suffices to show that we can find a good range assignment with high probability.

Fix $p < 1-\sqrt{2/3}$. Then, since $\theta_p \geq 1/2$, we can choose $\delta < 1/2$ so that $\theta_p > 1-\delta$. Choose positive constants α and $\beta < (1-2\delta)^2$ so that mapping Ψ_m applied to $\mathcal{B}_m\langle p\rangle$ gives a δ-dense $(O(m^{\log 6}), O(m^{\log \Gamma_\beta}))$-routable set on its leftmost level with probability at least $1-2^{-2^{\alpha m}}$ for large enough m.

Set $m = (1/\alpha)(\log n + 2)$ so that the probability of Ψ_m failing on a single m-stage subbutterfly of $\mathcal{B}_n\langle p\rangle$ is at most 2^{-4n}. We apply Ψ_m to these subbutterflies to obtain a range assignment. If Ψ_m yields a pair (U,V) for subbutterfly B, then we set $\rho_l(B) = U$ and, if B ends at one of the last $m+1$ levels, set $\rho_r(B) = V$. The probability that Φ_m fails at any of the subbutterflies is at most 2^{-3n} in total because there are fewer than 2^n such subbutterflies. If none fail, then we have a range assignment that satisfies conditions C1 and C2. Moreover, by an analysis similar to the one in the proof of Lemma 4.4, we can bound the probability to be at most 2^{-2n} that condition C3 is violated by the resulting range assignment ρ. Therefore, overall, with probability at least $1-2^{-n}$, the range assignment is $(O(m^{\log 6}), O(m^{\log \Gamma_\beta}), \delta, \beta)$-good for $\mathcal{B}_n\langle p\rangle$. Therefore by Lemma 5.2, with that probability we have an embedding with $O(1)$ load, $O(\log^{\log 6} n)$ dilation, and $O(\log^{\Gamma_\beta+1} n)$ congestion. □

Note that the embedding can be constructed by a deterministic algorithm with running time $N\log^{O(1)} N$, where N is the number of nodes of $\mathcal{B}_n$. This is because the mapping of each node or edge can be determined locally by looking at one or two subbutterflies consisting of $\log^{O(1)} N$ nodes. The time of such local computation, the main task of which is to compute the mapping Ψ_m, is polynomial in the size of the subbutterflies involved.

Using Theorem 3.1 instead of Theorem 4.1, we get the following theorem with better dilation although with exponentially worse congestion.

THEOREM 5.4. *For $p \leq 1-\sqrt{2/3}$, $\mathcal{B}_n\langle p\rangle$ admits a level-preserving self-embedding with $O(1)$ load, $O(\log n)$ dilation, and $n^{O(1)}$ congestion, with probability at least $1-2^{-n}$.*

6. A tight dilation lower bound for level-preserving embeddings. In this section, we prove a lower bound on the dilation of a level-preserving embedding in the faulty butterfly. The lower bound for $\mathcal{B}_n$ is $\log n - o(\log n)$, matching the upper bound of Theorem 5.4 up to a constant factor.

In fact, our lower bound is on a quantity called *drift* which in turn gives a lower bound for the dilation. We first define the drift of a path and then define the drift of a level-preserving embedding. Consider a path in $\mathcal{B}_n$ with an endpoint v at level i. The *drift* of the path from endpoint v is the maximum integer d such that the path contains a node on level $i-d$ or $i+d$. The *drift* of a path is the greater of the drifts of the path from its two endpoints. Finally, let Π denote the multiset of paths used in a level-preserving embedding, i.e., each single edge of the butterfly is mapped to a path of Π under the embedding. Then, the *drift* of the embedding is the maximum

of the drifts of the paths in Π. Note that the two endpoints of a path used in a level-preserving embedding always lie on consecutive levels. If such a path has drift d, then its length must be at least $2d-1$. Therefore, the dilation of a level-preserving embedding with drift d is at least $2d-1$.

Our strategy in proving the lower bound is to reduce the problem to the problem of *simple* embedding. We call a self-embedding of the butterfly *simple* if it is level-preserving and its dilation is one. We first show that, if a level-preserving self-embedding in $\mathcal{B}_n\langle p\rangle$ with drift d exists, then there must be a simple self-embedding in a smaller butterfly in which the distribution of faults depends on d as well as p. Then, we show that such a simple self-embedding is highly unlikely for small d.

Let i be in the range $d \le i \le n-d$, $x \in \{0,1\}^{i-d}$, and $y \in \{0,1\}^{n-i-d}$. The *d-disc on level i specified by x and y*, denoted by $D_d(x,y)$, is the set of nodes $\{\langle i, xwy\rangle \mid w \in \{0,1\}^{2d}\}$. Note that a d-disc on level i is the center level of a $2d$-stage subbutterfly. Also note that there is a path of drift at most d between every pair of vertices in a d-disc. The size of a d-disc is 2^{2d}.

We say two d-discs D and D' are *adjacent* if there exist $u \in D$ and $v \in D'$ such that u and v are adjacent in $\mathcal{B}_n$. Let $\mathcal{D}_{n,d}$ denote the graph where the nodes are d-discs of $\mathcal{B}_n$ and the adjacency is defined as above.

PROPOSITION 6.1. *$\mathcal{D}_{n,d}$ is isomorphic to $\mathcal{B}_{n-2d}$. More specifically, $\psi_{n,d} : \mathcal{B}_{n-2d} \to \mathcal{D}_{n,d}$ is an isomorphism if we set $\psi_{n,d}(\langle i, xy\rangle) = D_d(x,y)$ for each $i \in [0,\ldots,n-2d]$, $x \in \{0,1\}^i$, and $y \in \{0,1\}^{n-2d-i}$.*

Proof. Because x and y with $xy \in \{0,1\}^{n-2d}$ uniquely determines a d-disc $D_d(x,y)$, $\psi_{n,d}$ is clearly one-to-one. It is also onto because the numbers of nodes of the two graphs are the same. Now consider two adjacent nodes $u = \langle i, x_1 b x_2\rangle$ and $v = \langle i+1, x_1 b' x_2\rangle$ of $\mathcal{B}_{n-d}$, where $|x_1| = i$ and $b, b' \in \{0,1\}$. The d-disc $\psi_{n,d}(u) = D_d(x_1, bx_2)$ contains node $\langle i+d, x_1 b' 0^{2d-1} b x_2\rangle$ and the d-disc $\psi_{n,d}(v) = D_d(x_1 b', x_2)$ contains node $\langle i+d+1, x_1 b' 0^{2d-1} b x_2\rangle$; these nodes are adjacent and so are these discs. On the other hand, suppose d-discs $D_d(x_1,x_2)$ and $D_d(y_1,y_2)$, where $|x_1| = i$ and $|y_1| = i+1$, are adjacent in $\mathcal{D}_{n,d}$. Then, there is a node of the form $\langle i+d, x_1 x x_2\rangle \in D_d(x_1,x_2)$ and a node of the form $\langle i+d+1, y_1 y y_2\rangle \in D_d(y_1,y_2)$, which are adjacent in $\mathcal{B}_n$. This means that $x_1 x x_2$ and $y_1 y y_2$ may possibly disagree only at the $(i+d+1)$th bit. Therefore, x_1 is a prefix of y_1 and y_2 is a suffix of x_2; hence nodes $\langle i, x_1 x_2\rangle$ and $\langle i+1, y_1 y_2\rangle$ are adjacent in $\mathcal{B}_{n-2d}$. ∎

PROPOSITION 6.2. *If node u on level i, $d \le i < n-d$, and node v on level $i+1$ have a path of drift at most d connecting them, then the d-discs of u and v are adjacent.*

Proof. Simply note that any path from u with drift at most d keeps us in the rows that intersect with the d-disc of u. ∎

Now we can relate a level-preserving embedding of $\mathcal{B}_n$ with drift d to a simple embedding of $\mathcal{B}_{n-2d}$. For a fault set $F \in V(\mathcal{B}_n)$, let $\hat{F} \in V(\mathcal{B}_{n-2d})$ be defined by $\hat{F} = \{v \mid \psi_{n,d}(v) \subseteq F\}$, where $\psi_{n,d}$ is the isomorphism defined in Proposition 6.1. Intuitively, we regard a node in $\mathcal{B}_{n-2d}$ faulty if its corresponding d-disc is completely knocked out.

LEMMA 6.3. *Suppose there is a level-preserving embedding of $\mathcal{B}_n$ in $\mathcal{B}_n\langle F\rangle$ with load l and drift d. Then there is a simple embedding of $\mathcal{B}_{n-2d}$ in $\mathcal{B}_{n-2d}\langle \hat{F}\rangle$ with load at most $l2^{2d}$.*

Proof. Let ϕ be a level-preserving embedding of $\mathcal{B}_n$ in $\mathcal{B}_n\langle F\rangle$ with load l and drift d. Let ψ denote the standard isomorphism that maps $\mathcal{B}_{n-2d}$ to some fixed $(n-2d)$-stage subbutterfly of $\mathcal{B}_n$ starting at level d. We define a mapping ϕ' of nodes of $\mathcal{B}_{n-2d}$

to those of $\mathcal{B}_{n-2d}\langle\hat{F}\rangle$ by $\phi'(v) = \psi_{n,d}^{-1}(D)$, where $\psi_{n,d}$ is the isomorphism from $\mathcal{B}_{n-2d}$ to $\mathcal{D}_{n,d}$ and D is the d-disc that contains node $\phi(\psi(v))$. In other words, ϕ' maps v to a node that corresponds to the d-disc into which ϕ maps $\psi(v)$. Since ϕ avoids a d-disc completely contained in F, ϕ' avoids $\hat{F}$. Since each d-disc has 2^{2d} nodes, ϕ' maps at most $l2^{2d}$ nodes to a single node. Finally, by Propositions 6.1 and 6.2 it maps adjacent nodes to adjacent nodes. □

To prove our lower bound on the dilation of a level-preserving embedding, we use the unlikelihood of a simple embedding with small load. A simple embedding maps a monotone path to a monotone path preserving the levels of the endpoints. Therefore, the mapping of nodes on the outermost levels completely determines the embedding. We exploit this lack of flexibility of simple embeddings. In particular, we use the fact that the image of a simple embedding must contain a large number of binary trees rooted at level 0 which have an identical "shape" in the following sense.

Let h be an integer in $[n]$ and let P be a nonempty subset of $\{0,1\}^h$. A *P-tree of* $\mathcal{B}_n$ is a binary tree of height h contained in $\mathcal{B}_n$ such that

1. its root is on level 0; and
2. if the root is $\langle 0, xy\rangle$, where $|x| = h$, then the set of its leaves is $\{\langle h, zy\rangle \mid z \in P\}$.

Note that for each P there are exactly 2^n P-trees of $\mathcal{B}_n$, one for each node on level 0. These roots can be partitioned into 2^{n-h} subsets, the 2^h elements of each subset sharing a single set of leaves.

LEMMA 6.4. *Suppose there is a simple embedding of $\mathcal{B}_n$ in $\mathcal{B}_n\langle F\rangle$ with load l. Then, for every $h \in [n]$, there must be some $P \subseteq \{0,1\}^h$ with $|P| \geq 2^h/l$ such that the number of P-trees of $\mathcal{B}_n$ contained in $\mathcal{B}_n\langle F\rangle$ is at least $2^n/l$.*

Proof. Let ψ be a simple embedding of $\mathcal{B}_n$ in $\mathcal{B}_n\langle F\rangle$ with load l. Since ψ maps a monotone path to a monotone path, if u is a node on level 0 and v is a node on level n, then the monotone path from $\psi(u)$ to $\psi(v)$ does not intersect F. Let $U = \psi(Left(\mathcal{B}_n))$ and $V = \psi(Right(\mathcal{B}_n))$. Because the load of ψ is l, we have $|U| \geq 2^n/l$ and $|V| \geq 2^n/l$. Fix h and define $P = \{z \in \{0,1\}^h \mid \exists w \in \{0,1\}^{n-h} : \langle n, zw\rangle \in V\}$. Since $|V| \leq 2^{n-h}|P|$, we have $|P| \geq 2^h/l$ from the bound on V above. Let $u \in U$ and consider the P-tree T of $\mathcal{B}_n$ rooted at u. Since the monotone path from u to any leaf of T is the initial segment of some path from u to V, it must avoid F. Therefore T is contained in $\mathcal{B}_n\langle F\rangle$. Because this is true for any $u \in U$, the number of P-trees contained in $\mathcal{B}_n\langle F\rangle$ is at least $|U|$. □

Now we are ready to prove our lower bound.

THEOREM 6.5. *For any positive constants $p < 1$ and $\alpha < 1/3$, the probability is $1 - 2^{-\Omega(2^{n/3})}$ that any level-preserving embedding of $\mathcal{B}_n$ in $\mathcal{B}_n\langle p\rangle$ with load at most $2^{\alpha n}$ requires drift at least $(1/2 - o(1))\log n$.*

Proof. Let β be an arbitrary constant such that $0 < \beta < 1$. Suppose we have an embedding of $\mathcal{B}_n$ in $\mathcal{B}_n\langle F\rangle$ with load $2^{\alpha n}$ and drift at most $d = (\beta/2)\log n$. Then by Lemma 6.3, there is a simple embedding of $\mathcal{B}_m$ in $\mathcal{B}_m\langle\hat{F}\rangle$ with load at most $l = 2^{\alpha n}n^{\beta}$, where $m = n - 2d = n - \beta\log n$. This in turn implies, by Lemma 6.4, that for any $h \in [m]$, there exists a subset P of $\{0,1\}^h$ with $|P| \geq 2^h/l$ such that the number of P-trees contained in $\mathcal{B}_m\langle\hat{F}\rangle$ is at least $2^m/l$.

Set $h = n/3$. Let $F \in \mathbf{randsub}(V(\mathcal{B}_n), p)$. Then $\hat{F}$ is distributed according to $\mathbf{randsub}(V(\mathcal{B}_m), p^{n^\beta})$ because the size of a d-disc is n^β and d-discs are mutually disjoint. Call a P-tree *good* if it is contained in $\mathcal{B}_m\langle\hat{F}\rangle$. Our goal is to bound the probability that there exists $P \subseteq \{0,1\}^h$ with $|P| \geq 2^h/l$ such that the number of good P-trees is at least $2^m/l$. We start by estimating the probability of this event for a fixed $P \subseteq \{0,1\}^h$. We first claim that, for a fixed node u on level 0, the probability

that the P-tree rooted at u is good is at most $1/2l$ for sufficiently large n. This probability is simple to calculate:

$$\begin{aligned}(1-p^{n^\beta})^{2^h/l} &\le \exp[-p^{n^\beta}2^h/l] \\ &= \exp[-2^{h-\alpha n-o(n)}],\end{aligned}$$

but the negative exponent of this expression is exponential in $\Omega(n)$, while the exponent in $l = 2^{\alpha n}n^\beta$ is linear in n. Therefore this probability is definitely smaller than $1/(2l)$ for sufficiently large n. It follows that the expected number of P-trees contained in $\mathcal{B}_m\langle F\rangle$ is at most $2^m/(2l)$. To obtain a tail estimate, we partition the set of 2^m nodes on level 0 of $\mathcal{B}_m$ into 2^h subsets $U_x = \{\langle 0, xy\rangle \mid y \in \{0,1\}^{m-h}\}$, $x \in \{0,1\}^h$. Then, for each fixed $x \in \{0,1\}^h$, the P-trees rooted at nodes in U_x do not intersect each other and hence the events that they are contained in $\mathcal{B}_m\langle \hat{F}\rangle$ are independent of each other. Therefore, we can apply the partition Chernoff bound (Proposition 2.3): the probability that at least $2^m/l$ nodes on level 0 are roots of good P-trees is

$$\begin{aligned}2^{-\Omega(2^{m-h}/l)} &= 2^{-\Omega(2^{(2/3-\alpha)n-o(n)})} \\ &= 2^{-\Omega(2^{\gamma n})},\end{aligned}$$

where γ is a constant $1/3 < \gamma < 2/3 - \alpha$.

Now consider all possible subsets $P \subseteq \{0,1\}^h$. There are at most 2^{2^h} such subsets to consider so that the probability that there exist any P with the required property is at most $2^{2^h-\Omega(2^{\gamma n})} = 2^{-\Omega(2^{\gamma n})}$. □

7. Open questions. Although our upper bound easily transfers to the CCC, it is not clear if the result can be extended to the shuffle-exchange or the de Bruijn network, which belong to the other type of hypercubic bounded-degree networks. For the butterfly itself, although we have a tight bound for the dilation of level-preserving embeddings, no nontrivial lower bound is known for general embeddings. We might also ask for the best possible load/dilation/congestion ever achievable by any bounded-degree network.

Recently, Karlin, Nelson, and Tamaki [9] have shown the existence of a critical probability for the presence of a linear-sized connected component in the random subgraph of the butterfly. In their edge fault model, where each edge fails independently with probability p, the critical probability p_c is shown to be between 0.564 and 0.663; for $p < p_c$ the faulty butterfly contains a component with $\Omega(N)$ nodes with high probability, whereas for $p > p_c$ it lacks such a component with high probability. A similar critical probability can be obtained for our node fault mode. It is an interesting question whether the embedding results of this paper can be extended to every node-failure probability smaller than the critical probability.

Even more recently, Cole, Maggs, and Sitaraman [5] have provided yet more evidence that the butterfly network is highly fault tolerant: in a faulty butterfly with some constant node-failure probability, it is possible to identify a set of $\Theta(N)$ nodes among which any permutation routing is performed in $O(n)$ steps with high probability. Constant-factor slowdown simulation of the butterfly by such a faulty butterfly, however, is still open.

Acknowledgments. I would like to thank Allan Borodin for valuable comments on this work, Nick Pippenger for his suggestion of the use of branching processes in tackling the problem, and Tino Tamon for reading an earlier manuscript and making useful comments.

REFERENCES

[1] B. Aiello and F. T. Leighton, *Coding theory, hypercube embeddings, and fault tolerance*, in Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, 1991, pp. 125–136.

[2] N. Alon and J. H. Spencer, *The Probabilistic Method*, John Wiley and Sons, New York, 1992.

[3] F. Annexstein, *Fault tolerance in hypercube-derivative networks*, in Proceedings of the 1st Annual ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, 1989, pp. 179–188.

[4] K. B Athreya and P. E. Ney, *Branching Processes*, Springer-Verlag, Berlin, 1972.

[5] R. Cole, B. Maggs, and R. Sitaraman, *Routing on butterfly networks with random faults*, in Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 558–570.

[6] J. Håstad, F. T. Leighton, and M. Newman, *Fast computation using faulty hypercubes*, in Proceedings of the 21st Annual ACM Symposium on Theory of Computing, ACM, New York, 1989, pp. 251–263.

[7] C. Kaklamanis, A. R. Karlin, F. T. Leighton, V. Milenkovic, P. Raghavan, S. Rao, C. Thomborson, and A. Tsantilas, *Asymptotically tight bounds for computing with faulty arrays of processors*, in Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 285–296.

[8] A. Karlin and G. Nelson, *On the Existence of the Ocean in a Faulted Butterfly*, unpublished note, 1989.

[9] A. Karlin, G. Nelson, and H. Tamaki, *On the fault tolerance of the butterfly*, in Proceedings of the 26th Annual ACM Symposium on Theory of Computing, ACM, New York, 1994, pp. 125–133.

[10] R. Koch, T. Leighton, B. Maggs, S. Rao, and A. Rosenberg, *Work-preserving emulations of fixed-connection networks*, in Proceedings of the 21st Annual ACM Symposium on Theory of Computing, ACM, New York, 1989, pp. 227–240.

[11] F. T. Leighton, B. Maggs, and R. Sitaraman, *On the unexpected fault-tolerance of some popular bounded-degree networks*, in Proceedings of the 33d Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 542–552.

[12] F. T. Leighton, B. Maggs, and S. Rao, *Packet routing and job-shop scheduling in O(congestion + dilation) steps*, Combinatorica, 14 (1994), pp. 167–186.

[13] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, MA, 1995.

[14] N. Pippenger, *The asymptotic optimality of spider-web networks*, Discrete Appl. Math., 37/38 (1992), pp. 437–450.

[15] H. Tamaki, *Efficient self-embedding of butterfly networks with random faults*, in Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 533–541.

[16] H. Tamaki, *Robust bounded-degree networks with small diameters*, in Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, 1992, pp. 247–256.

SIAM J. COMPUT.
Vol. 27, No. 3, pp. 637–653, June 1998
© 1998 Society for Industrial and Applied Mathematics
003

SPLITTINGS, ROBUSTNESS, AND STRUCTURE OF COMPLETE SETS*

HARRY BUHRMAN†, ALBRECHT HOENE‡, AND LEEN TORENVLIET§

Abstract. We investigate the structure of *EXP*-complete and hard sets under various kinds of reductions. In particular, we are interested in the way in which information that makes the set complete is stored in the set. We study for various types of reductions the question of whether the set difference $A - S$ for a hard set A and a sparse set S is still hard. We also address the question of which complete sets A can be split into sets A_1 and A_2 such that $A \equiv_r^P A_1 \equiv_r^P A_2$ for reduction type r, i.e., which complete sets are *mitotic*. We obtain both positive and negative answers to these questions depending on the reduction type and the structure of the sparse set.

Key words. reductions, mitoticity, completeness, complexity

AMS subject classifications. 03D15, 03D20, 68Q15

PII. S0097539795279724

1. Introduction.

The structure of complete sets under various types of reductions is a well-studied subject in complexity theory. The question "Which sets can be complete under which type of reductions?" has been posed many times and answered for many complexity classes. (See [9] for an overview.)

A complete set represents, through the reduction under which it is complete, an entire complexity class. A membership algorithm for the complete set combined with a reduction (of the appropriate type) gives a membership algorithm for a set in the class. Viewed as such, the complete set contains the information of any particular set in the class.

In this paper we investigate the structure of the information that makes the set complete for deterministic exponential time and for various types of reductions. We take an approach pioneered by Schöning [21]. To investigate the structure of a complete set we compare this set with another set. Schöning in particular showed that for a $\leq_m^P$-complete set A in *EXP* and a polynomial time computable set D, the set $A\Delta D$ is of exponential density. Tang, Fu, and Liu took in [23] the approach of taking the difference of sets complete in exponential time with a sparse subexponential time computable (sub)set and asked the question of whether the resulting set is complete. In section 3, we extend their work by studying analogous questions for other types of completeness. We obtain similar results and show that for several types of reductions and arbitrary sparseness conditions there exists a *single* subexponential time computable set S that meets this sparseness condition and, furthermore, has the property that for any set A, complete under the reduction for which this S is constructed, the set $A - S$ is no longer complete. In addition, it seems possible to make S "almost

*Received by the editors January 11, 1995; accepted for publication (in revised form) February 19, 1996.

http://www.siam.org/journals/sicomp/27-3/27972.html

†Centrum voor Wiskunde en Informatica, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands (buhrman@cwi.nl). The research of this author was partially supported by the NWO through NFI Project ALADDIN number NF 62-376.

‡Department of Computer Science, Technische Universität Berlin, D-1000 Berlin 10, Germany. The research of this author was partially supported by Deutsche Forschungsgemeinshaft, Postdoktorandenstipendium.

§Department of Mathematics and Computer Science, University of Amsterdam, Plantage Muidergracht 24, 1018 TV Amsterdam, The Netherlands (leen@wins.ua.nl).

polynomial time computable." It follows from the constructions that, by taking a sufficiently slow enumeration of reductions, we can lower the time complexity of S to any reasonably behaved superpolynomial function. We also address the flip question in this section: "For which reductions and which sparse sets is it the case that the set $A - S$ is complete?" We study sparse sets S of simple *structure* rather than computationally simple sets. Selman [22] introduced p-selective sets as a resource bounded analog of semirecursive sets introduced by Jockusch [13]. For any tally set, a polynomial time Turing equivalent p-selective set can be found. Therefore p-selective sets can be computationally very complex. Nonetheless, p-selective sets are intuitively easy to compute since, for any two strings x and y, it can be decided in polynomial time which of the two is more likely (or actually no less likely) to be in the set. It turns out that for most of the reductions studied in subsection 3.1 the set $A - S$ remains complete if S is a p-selective set. We prove this theorem for disjunctive, conjunctive, and 2-truth-table (2-*tt*) reductions. One might expect that, as is the case with many results on reductions, our 2-truth-table theorem can be extended to at least bounded truth-table reductions. From a recent result of Buhrman, Fortnow, and Torenvliet [6] it follows, however, that the 2-*tt* result cannot be extended even to 3-*tt*.

In section 4 we study sets that remain complete even when another *complete* set is removed, i.e., we study sets that can be "split" into two or more sets that are again complete. Such sets have been studied in recursion theory and are called *mitotic* sets. We follow the line of Ambos-Spies [1] and prove that $\leq_m^P$-complete sets for *EXP* indeed are (weakly p-m) mitotic. In contrast, we show that there exists a $\leq_{3\text{-}tt}^P$-complete set that is not weakly p-m mitotic. Finally, we show a counterpart to Ladner's splitting theorems [16], i.e., we construct a set that can be split into two parts that are strictly below the degree of complete sets and that are $\equiv_m^P$ instead of incomparable.

2. Definitions and notation. We assume that the reader is familiar with standard notions in structural complexity theory as they are defined, e.g., in [2]. All kinds of *polynomial-time bounded* reductions—many-one, (disjunctive and conjunctive) truth-table and Turing—are frequently used without explanation. We use the following notation for the (polynomial-time computable versions) of the different types of reductions: $\leq_m^P$ for many-one reductions, $\leq_1^P$ for one-one reductions, $\leq_{k\text{-}tt}^P$ for k-truth-table reductions, $\leq_{btt}^P$ for bounded truth-table reductions, $\leq_{tt}^P$ for truth-table reductions, $\leq_c^P$ for conjunctive truth-table reductions, $\leq_d^P$ for disjunctive truth-table reductions, $\leq_{k-d}^P$ for k-disjunctive truth-table reductions, and $\leq_T^P$ for Turing reductions. The symbols $\leq_m$ and $\leq_T$ also appear in the paper, without the superscript P, to indicate the version of these reductions without time bound.

Various definitions for these types of reductions can be found in the literature, e.g., in [17, 7, 5, 8]. We think of polynomial-time bounded reductions as being modeled by adaptive and nonadaptive oracle machines. We use various enumerations $\{M_i\}_i$ of (oracle) Turing machine programs with varying properties. If the type of machine is not clear from the context, we explicitly mention the machine type. An enumeration $\{M_i\}_i$ can thus stand for an enumeration of all polynomial-time bounded machines in one theorem and an enumeration of all bounded truth-table reductions in the next. In the case of enumeration of many-one reductions, i.e., where the machines are transducers, we also use $\{f_i\}_i$ to emphasize this fact. For polynomial-time bounded machines we always assume machine M_i in such an enumeration to be time bounded by $n^i + i$, where n is the length of the input. Usually, we denote the set of strings queried on input x by machine i with oracle A by $Q_i^A(x)$, or by $Q_i(x)$ if M_i is non-adaptive.

The result of the computation (accept/reject or the value computed) of machine M_i on input x (relative to oracle A) is sometimes denoted as $M_i(x)$ ($M_i^A(x)$), where $M_i(x) = 0$ or 1 means that the computation rejects or accepts, respectively.

Sets of strings are denoted by capital letters and are subsets of Σ^*, where $\Sigma = \{0, 1\}$. Strings are denoted as small letters $x, y, u, v, \ldots$. The length of a string x is denoted by $|x|$.

We assume *pairing and projection functions* that are easy to compute. For strings, the pairing of $x_1, \ldots, x_n$ is denoted by $<x_1, \ldots, x_n>$, and $\pi_i(y)$ is the projection of y onto its ith coordinate. We assume all kinds of convenient properties of these pairing functions, e.g., pairing functions can be designed such that $<x, y> < <x, z>$ whenever $y < z$ or such that $|<x, z>| = |<x, z'>|$ for a large (exponential in this length) number of pairs z and z'. In fact, the specific properties of different pairing functions may depend on the context of the proof in which they are used. Integer numbers can also appear as arguments to pairing functions. If so, the integer is identified with its binary representation.

An ordering on Σ^* is assumed where $x < y$ if $|x| < |y|$ and that coincides with the lexicographical ordering if $|x| = |y|$. The cardinality of a set A is denoted as $\|A\|$. The value of the *characteristic function* of a set A on a string x is denoted by $\chi_A(x)$, i.e., $\chi_A(x) = 1$ if $x \in A$ and 0 otherwise. Following Kelly [14], we let the notation $\bigcup\{S : S \text{ meets condition }\}$ stand for the union of all sets S that meet the given condition. For sets A and B the notation $A \oplus B$ stands for the disjoint union of A and B, i.e., $\{0x : x \in A\} \cup \{1x : x \in B\}$.

For a set A

- for $n \in \omega$, we let the notation $A^{\leq n}$ stand for the set consisting of all strings in A of length $\leq n$; and
- for a string x, we let $A^{[x]}$ stand for the x *section of* A, i.e., the set $\{<y, z> : y = x \text{ and } <y, z> \in A\}$.
- in order to measure the density of A, we say that A is $g(n)$ sparse for some nondecreasing total function $g : \omega \to \omega$, if for all n, $\|A^{\leq n}\| < g(n)$.

The main complexity classes considered in this paper are P (polynomial time), EXP (exponential time), and $NEXP$ (nondeterministic exponential time). For the latter classes, we allow polynomials to act as exponents in the time bounds, e.g., $EXP = \bigcup\{DTIME(2^{n^i}) : i \in \omega\}$. For EXP, the set K is the universal complete set. $K = \{<i, x, l> : M_i \text{ accepts } x \text{ in} \leq l \text{ steps }\}$.

3. The robustness of completeness notions for exponential time. In this section we study the question of which sparse sets can be removed from exponential-time complete sets of different types without disturbing the completeness of these sets. The question originates from work of Schöning [21], who showed that for every $\leq_m^P$-hard set A for EXP and every set D in P, the set $A \Delta D$ is of exponential density. In [23], Tang, Fu, and Liu showed, as a corollary to an analogous result on parity reductions, that even for subexponential time computable D, the difference $A \Delta D$ remains $\leq_m^P$-hard for exponential time.

They further show that for an arbitrary sparseness condition, there exists a single subexponential time computable set S, such that for *any* exponential time complete set A, the set $A - S$ is no longer exponential time hard. Their proof hinges on the fact that for any exponential time computable set B and any exponential time complete set A, there exists a *length increasing* reduction from B to A. Subsequently, the subexponential time computable set is constructed by choosing a sufficiently sparse polynomial-time computable subset of $\{0\}^*$ and defining S as the image of this set

varying over all polynomial-time computable functions, i.e., $S = \{0^{b_i} : |f_i(0^{b_i})| > b_i\}$, where b_i are chosen sufficiently far apart.

A closer look at the proof shows although the theorem just states that S is subexponential time computable, that there are various ways of making S come arbitrarily close to polynomial time. It therefore seems reasonable to ask whether we can also choose S to be polynomial-time computable. The answer to this question is negative, as observed in [23]. From the $\leq_1^P$-reduction of $K' = K \times \Sigma^*$ to the *EXP* complete set A, we can easily construct a $\leq_m^P$-reduction to $A - S$ for any polynomial-time computable sparse set S. In this section we will investigate, for different types of reductions, which sparse sets can destroy the completeness of a given set, and for which sparse sets completeness is preserved.

3.1. Sparse sets that destroy completeness. The set K' defined above is, of course, $\leq_m^P$-complete for *EXP*. In fact it is $\leq_d^P$-complete for *EXP* in a special way. For a given string x either all strings $<x, y>$ are in this set, or all are out depending on $x \in K$. Therefore, as long as S is $p(n)$ sparse, the set $K' - S$ remains $\leq_d^P$-complete for *EXP*. The reduction from K to K' on input x queries just the set $\{<x, y> | 0 \leq y \leq p(2n) + 1\}$. Since all of these strings have length $\leq 2|x|$, at least one of them is not in S and it is in K' iff $x \in K$. This explains why the theorem "*there exists a sparse set S, such that for any* $\leq_d^P$*-complete set A for EXP, the set* $A - S$ *is not* $\leq_d^P$*-complete for EXP*" cannot exist.

The best we can hope for is a theorem for reductions that can query at most $\|S^{\leq n}\|$ strings for each length n. Since we want the construction of S to meet any given sparseness condition, this implies that the number of queries cannot be a function of n that grows to infinity with n. In other words, the number of queries must be some fixed constant. Such reductions are called *bounded truth-table reductions*, and for these reductions we can obtain the theorem. In fact, since the proof method of our theorem is not dependent on the reductions being nonadaptive, we can obtain the theorem for bounded Turing reductions ($\leq_{bT}$).

THEOREM 3.1. *Let g be a recursively computable nondecreasing function with $\lim_{n\mapsto\infty} g(n) = \infty$, and let f be a function that is superpolynomial, subexponential, and time constructible. There exists a $g(n)$-sparse $O(f(n))$-time computable set S such that for any $\leq_{btt}^P$-complete set A for EXP, the set $A - S$ is no longer $\leq_{btt}^P$-complete.*

Proof. For a given set A, we demonstrate the existence of a set L_A such that $L_A \leq_{btt}^P A$ iff there exists a reduction M_i that queries at least one string $y \in A$ with $|y| > b(i)$ on input $0^{b(i)}$. Next, we let all such strings be elements of S, thereby preventing the existence of such a reduction from L_{A-S} to $A - S$ and hence completeness of $A - S$. This technique was borrowed from Watanabe [25].

Let $\{M_i\}_i$ be an enumeration of all $\leq_{btt}^P$-reductions. Without loss of generality, we may assume that machine M_i generates $\leq i$ queries on any input. First we define a set of numbers $\{b(i)\}$ sufficiently far apart and sufficiently easy to recognize, i.e., we want that, for each n, the question "$\exists i : n = b(i)$?" can be answered in time $O(n)$, and furthermore we want for each i that $g(b(i)) > i^2$. This means that we can define at least i^2 strings in $S^{\leq b(i)}$ without disturbing the sparseness condition on S. In fact, we will put just $\leq i$ strings in each interval $S^{\leq b(i+1)} - S^{\leq b(i)}$. Finally, we wish to compute $Q_i(0^{b(i)})$ to define strings in S. To be able to do this in time $f(b(i))$, we need that $2^{b(i)} > f(b(i)) > (b(i))^i + i$, for all i.

We let $S = \{y : \exists i[y \in Q_i(0^{b(i)}) \wedge |y| > b(i)]\}$ and claim that S is the set searched for. First, $\|S^{\leq n}\| \leq g(n)$ for each n, since for each n the only strings that are in $S^{\leq n}$ are in $\bigcup\{Q_j(0^{b(j)}) : j \leq i\}$ for i maximal such that $b(i) \leq n$. Hence, there are at

most i^2 strings in $S^{\leq n}$, and $b(i)$ was chosen such that $g(n) \geq g(b(i)) > i^2$ as required. Next, S is $O(f(n))$-time computable by the choice of $b(i)$. Finally, the set $A - S$ is not $\leq^P_{btt}$-complete. To see this, assume that $A - S$ is $\leq^P_{btt}$-complete and define the set $L_{A-S} = \{0^{b(i)} : M_i^{(A-S)^{\leq b(i)}}(0^{b(i)}) = 0\}$. L_{A-S} is exponential time computable since $2^{b(i)} > b(i)^i + i$ and since both A and S are exponential time computable and are only queried on inputs of length $\leq b(i)$ on input $0^{b(i)}$. It follows that $L_{A-S} \leq^P_{btt} A - S$. Let M_j be the reduction from L_{A-S} to $A - S$. Then $Q_j(0^{b(j)}) \cap (A - S) \subseteq \Sigma^{\leq b(j)}$ and therefore $0^{b(j)} \in L_{A-S}$ iff $M_j^{A-S}(0^{b(j)})$ rejects, which is a contradiction. □

If we change S in the construction above to $\{\min\{y : y \in Q_i(0^{b_i}) \wedge |y| > b_i\} : i \in \omega\}$, we get the following corollary from the same construction.

COROLLARY 3.2. *Given a recursively computable nondecreasing function $g(n)$ with $\lim_{n\mapsto\infty} g(n) = \infty$, there exists a $g(n)$-sparse subexponential time computable set S such that for any $\leq^P_c$-complete set A for EXP the set $A - S$ is no longer $\leq^P_c$ complete.*

Conjunctive truth-table (c-*tt*) reducibilities form an exception in yet another way. For these reductions we can even let the set A be *EXP* hard instead of *EXP* complete. We use the fact that for conjunctive truth-table reductions we can get a kind of 1-1 behavior for the query sets. A similar result for $\leq^P_m$-hard sets that uses the fact that these sets are also hard under $\leq^P_1$-reductions appears in [23]. We isolate and prove this property in the following lemma.

LEMMA 3.3. *If A is $\leq^P_c$-hard for EXP, then for any set B in EXP there exists a $\leq^P_c$-reduction M_j from B to A such that $Q_j(x) \not\subset \bigcup\{Q_j(y) : y \in B \wedge y < x\}$.*

Proof. Let $\{M_i\}_i$ be an enumeration of $\leq^P_c$-reductions We construct a set W as follows. On input $<i, x>$ compute $Q_i(<i, x>)$. If $Q_i(<i, x>) \subseteq \bigcup\{Q_i(<i, y>) : <i, y> < <i, x> \wedge <i, y> \in W\}$, then we let $<i, x> \notin W$; otherwise $<i, x> \in W$ iff $x \in B$.

It is easy to see that W is in *EXP*, so there exists a $\leq^P_c$-reduction from W to A, say, M_j. For this reduction it follows that $Q_j(<j, x>) \not\subset \bigcup\{Q_j(<j, y>) : <j, y> \in W \wedge <j, y> < <j, x>\}$. By assumption on the pairing function that $<i, x> < <i, y> \leftrightarrow x < y$ the function $M'(x) = M_j(<j, x>)$ computes a $\leq^P_c$-reduction with the required property. □

From this lemma we get the following theorem.

THEOREM 3.4. *Let $g(n)$ be a recursively computable nondecreasing function with $\lim_{n\mapsto\infty} g(n) = \infty$. There exists a $g(n)$-sparse set S in EXP such that for any $\leq^P_c$-hard set A for EXP the set $A - S$ is no longer $\leq^P_c$-hard.*

Proof. Again, we let the numbers $b(i)$ be sufficiently far apart to guarantee $g(n)$ sparseness of S if we put one string in S for each $b(i)$ and such that $0^{b(i)}$ is again easy to recognize. Furthermore, we let $2 \times (b(i-1))^{(i-1)} < b(i)$ to avoid confusion later on. Then we put the least string in $\bigcup\{Q_i(<0^{b(i)}, y>) : |y| \leq b(i)\}$ of length $\geq b(i)/2 - 1$ in S. (If no such string exists we do nothing.)

S is exponential time computable since, to decide membership of a string x in S, we search for a $0^{b(i)}$ such that $b(i) \leq 2 \times |x|$ and $|x| < (b(i))^i$. (There can be only one candidate.) Now compute the query sets on the, at most, $2^{2|x|}$ different y in time $\leq 2^{2|x|} \times (2|x|)^i + i$, and see if x is the least string of the right length in the union of these sets.

$A - S$ is not $\leq^P_c$-hard. If it were, then one of the $\leq^P_c$-reductions from $D = \{<x, y> : x \in \{0\}^*, y \in \Sigma^*\}$ to A would behave as predicted by Lemma 3.3.

Let M_j be such a reduction. It follows that $\|\bigcup\{Q_j(<0^{b(j)}, y>) : |y| \leq b(j)\}\| \geq 2^{b(j)}$ from Lemma 3.3. Hence there is one string in this set of length $\geq b(j)/2 - 1$.

The least of these strings z_0 is not in $A - S$ by construction, and M_j must reject any input $<0^{b(j)}, y>$ for which z_0 is in $Q_j(<0^{b(j)}, y>)$. Yet $<0^{b(j)}, y> \in D$, so M_j cannot be a reduction from D to $A - S$. □

Conjunctive and disjunctive truth-table reducibilities are kind of each other's complement. If a set is conjunctive truth-table reducible to a set A, then its complement is disjunctive truth-table reducible to $\overline{A}$. So we find the following.

COROLLARY 3.5. *Let $g(n)$ be a recursively computable nondecreasing function with $\lim_{n\mapsto\infty} g(n) = \infty$. There exists a $g(n)$-sparse subexponential time computable set S such that for any $\leq_d^P$-complete set A for EXP the set $A \cup S$ is no longer $\leq_d^P$-complete.*

In addition we have the following.

COROLLARY 3.6. *Let $g(n)$ be a recursively computable nondecreasing function with $\lim_{n\mapsto\infty} g(n) = \infty$. There exists a $g(n)$-sparse set S in EXP such that for any $\leq_d^P$-hard set A for EXP the set $A \cup S$ is no longer $\leq_d^P$-hard.*

Proof. If $K \leq_c^P A$ via M_i, then

$$\begin{array}{rrcl} & x \in K & \Leftrightarrow & Q_i(x) \subset A, \\ \text{or} & x \notin K & \Leftrightarrow & Q_i(x) \cap \overline{A} \neq \emptyset. \end{array}$$

Now $\overline{K} \in EXP$ so $\overline{K} \leq_m^P K$, say, via f.

$$\begin{array}{rrcccl} \text{So} & x \in \overline{K} & \Leftrightarrow & f(x) \in K & \Leftrightarrow & Q_i(f(x)) \subset A, \\ \text{or} & x \notin \overline{K} & \Leftrightarrow & f(x) \notin K & \Leftrightarrow & Q_i(f(x)) \cap \overline{A} \neq \emptyset, \\ \text{or} & x \in K & \Leftrightarrow & f(x) \notin K & \Leftrightarrow & Q_i(f(x)) \cap \overline{A} \neq \emptyset, \end{array}$$

so $\overline{A}$ is $\leq_d^P$-hard via $M_i(f(x))$. Along the same lines, if A is $\leq_d^P$-hard then $\overline{A}$ is $\leq_c^P$-hard. So A is $\leq_d^P$-complete (hard) iff $\overline{A}$ is $\leq_c^P$-complete (hard) for EXP. But if $\overline{A}$ is $\leq_c^P$-complete (hard), then there exists a $g(n)$-sparse subexponential (exponential) time computable set S such that $\overline{A} - S$ is no longer $\leq_c^P$-complete (hard), and then $A \cup S$ is no longer $\leq_d^P$-complete (hard). □

3.2. Easy sparse sets. As in the case of $\leq_m^P$-complete sets, we can let the time complexity of the set S in the previous subsection come arbitrarily close to polynomial time. For some—but surprisingly not all—reductions, a polynomial-time computable sparse set that destroys completeness does not exist. To show this we take a slightly more general view of the complexity of the set S. Instead of taking S polynomial-time computable, we let S be p-selective. p-selective sets were introduced by Selman [22] as a resource bounded analogue of semirecursive sets, which were introduced by Jockusch [13].

DEFINITION 3.7. *A set A is called* p-selective *iff there exists a polynomial-time computable function $f : \Sigma^* \times \Sigma^* \mapsto \Sigma^*$, called a* p-selector, *such that for any $x, y \in \Sigma^*$*

1. *$f(x, y) \in \{x, y\}$, and*
2. *$\chi_A(f(x, y)) = \max\{\chi_A(x), \chi_A(y)\}$.*

The following "ordering lemma" is used in the proofs of the theorems in this section and can be found (in various forms) in, e.g., [24, 10, 11].

LEMMA 3.8. *Let $V = \{v_1, \ldots, v_n\}$ be a finite set of strings, and let A be a p-selective set with p-selector f. The strings in V can be ordered "according to f" as $v_{i_1}, \ldots, v_{i_n}$ such that $v_{i_j} \in A \to v_{i_{j+1}} \in A$ in time polynomial in $|v_1| + \cdots + |v_n|$.*

As only polynomial time is involved, we sometimes assume a finite set "ordered according to a p-selector," without loss of generality.

For any (nonfinite) sparseness condition a p-selective set (which may be infinite) that meets this condition can be constructed. For polynomially sparse p-selective

sets we can show counterparts to the theorems of subsection 3.1. For conjunctive, disjunctive, and 2-*tt* reductions, for a set A that is complete under this reduction, and for a set S that is polynomially sparse and p-selective, the set $A - S$ remains complete under the same reduction. Since all sets in P are p-selective, the same results follow for sparse polynomial-time computable sets.

It might be instructive here to compare our results to those of Tang, Fu, and Liu [23, Theorems 3.6 and 3.7]. They show that sets complete under conjunctive and disjunctive reductions stay complete when a sparse subexponential-time computable *subset* is removed. Our sparse set may be *any* set.

THEOREM 3.9. *For any set A that is $\leq_c^P$-hard for EXP and any $p(n)$-sparse p-selective set S, the set $A - S$ remains $\leq_c^P$-hard for EXP.*

Proof. Let f be a p-selector for S. We construct an exponential time-computable set W such that, for some index w of a conjunctive truth-table reduction M_w from W to A, it holds that the universal set K is many-one reducible to $W^{[w]}$ and $W^{[w]}$ is in turn conjunctive truth-table reducible to $A - S$. This then establishes completeness of $A - S$.

The set W consists of strings $<i, x, z>$ for $i \in \omega$, $x \in \Sigma^*$, and z ranging from $0^{|x|}$ through $1^{|x|}$. Fix i and x and let $n = |<i, x, x>|$. By appropriate assumptions on the pairing function, all of the pairs $<i, x, z>$ are of the same length. So, since S is $p(n)$ sparse, there exists some polynomial q such that at most $q(n)$ of the different strings queried on such an input can be in S. Let Z be the set of the lexicographically first $q(n)+1$ strings in the interval $0^{|x|}$ through $1^{|x|}$. We describe membership of $<i, x, z>$ in W. If $z \notin Z$, then $<i, x, z> \notin W$. Let $U = \bigcup\{Q_i(<i, x, z>) : z \in Z\}$. Assume that $U = \{u_1, u_2, \ldots, u_m\}$ such that $u_i \in S \rightarrow u_{i+1} \in S$ and let $U' = \{u_{m-q(n)+1}, \ldots, u_m\}$. If $U = \emptyset$ or time 2^n is insufficient to compute U and sort U according to the p-selector, then no string $<i, x, z>$ is in W. It will follow from the construction that the first case cannot occur if i is a program that computes a c-*tt* reduction from W to A. The latter case can only occur for finitely many different x.

Now there are two cases.

Case 1. There is a $z \in Z$ such that $Q_i(<i, x, z>) \subseteq \bigcup\{Q_i(<i, x, z'>) : z' \in Z \wedge z' \neq z\}$. Let z_0 be the least such z. We let $<i, x, z> \in W \iff z \neq z_0$.

Case 2. There is no such z. Then $\|Q_i(<i, x, z>) - \bigcup\{Q_i(<i, x, z'>) : z' \in Z \wedge z' < z\}\| \geq 1$ for all $z \in Z$ and hence, there is a z such that $Q_i(<i, x, z>) - U' \neq \emptyset$ *and* $Q_i(<i, x, z>) \cap U' \subseteq \bigcup\{Q_i(<i, x, z'>) : z' \in Z \wedge z' < z\}$. Let z_0 be the least such z. We let $<i, x, z_0> \in W \leftrightarrow x \in K$. We let $<i, x, z> \in W$ for all $z < z_0$ in Z and let $<i, x, z> \notin W$ for all $z > z_0$ in Z.

This ends the construction of W.

Since W is computable in exponential time, there is a conjunctive truth-table reduction from W to A. Let M_w be such a reduction. We show a reduction from $W^{[w]}$ to $A - S$. Let $<w, x, z>$ be some input.

1. As mentioned above, there is a finite number of cases for which the string $<w, x, z>$ is not in W because of insufficient computation time. In these cases, or in the case where z is not among the first $q(|<w, x, x>|)+1$ strings, our reduction rejects. (Or, equivalently, produces some fixed string not in $A - S$.)
2. Otherwise, compute the set $U = \{u_1, \ldots, u_m\}$ as described above. (Since w is a fixed constant, this can be done in polynomial time in $|<w, x, z>|$.) Since M_w is a reduction from W to A it follows that the construction falls in Case 2. If z is the least string in Z satisfying Case 2, then produce $Q_w(<w, x, z>) - U'$. Otherwise accept or reject according to Case 2.

This reduction accepts $<w, x, z>$ iff $<w, x, z> \in W^{[w]}$ and queries only strings in $A - S$.

The reduction from K to $W^{[w]}$ is the following. If x belongs to the finite number of exceptional cases, then we find out by table-lookup if $x \in K$ and accept or reject accordingly. Otherwise, we compute U for this w and x and the least z satisfying Case 2 and let $f(x) = <w, x, z>$. □

In the proof of this theorem we defined a set W such that the reduction of W to A must query strings that are in A on certain inputs. We did this by defining certain strings in W (hence a conjunctive reduction must get the answer yes on queries produced on this input). On the other hand, the reduction of W to A must query some strings outside S on some inputs. We did this by diagonalizing against reductions for which the cardinality of the union of the query sets on the inputs $<i, x, z>$ was small enough to "fit" inside S. Then a reduction from any exponential time-computable set to $A - S$ can be constructed, using W, by assuming that queries that may be in S are answered yes, and querying the remaining ones.

For $\leq_d^P$-hard sets, we can change the strategy of the proof to construct a set W such that we know that queries that may be in S are answered *negatively* (by leaving enough strings *out* of W). A completely analogous proof then yields the following.

THEOREM 3.10. *For any set A that is $\leq_d^P$-hard for EXP and any $p(n)$-sparse p-selective set S, the set $A - S$ remains $\leq_d^P$-hard for EXP.*

By complementation Theorems 3.9 and 3.10 yield the following corollaries, respectively.

COROLLARY 3.11. *For any set A that is $\leq_d^P$-hard for EXP and any $p(n)$-sparse p-selective set S, the set $A \cup S$ remains $\leq_d^P$-hard for EXP.*

COROLLARY 3.12. *For any set A that is $\leq_c^P$-hard for EXP and any $p(n)$-sparse p-selective set S, the set $A \cup S$ remains $\leq_c^P$-hard for EXP.*

We can also prove the same theorem for $\leq_{2\text{-}tt}^P$-hard sets.

THEOREM 3.13. *Let A be $\leq_{2\text{-}tt}^P$-hard for EXP and let S a p-selective $p(n)$-sparse set. The set $A - S$ is still $\leq_{2\text{-}tt}^P$-hard for EXP.*

Proof. We will construct an exponential time-computable set W that is used in defining a $\leq_{2\text{-}tt}^P$-reduction from K to $A - S$. Let f be the p-selector for S. W consists of strings $<i, x, z>$, where $|z| = |x|$.

Fix some i and x. Let $n = |<i, x, x>|$. We will define membership of $<i, x, z>$ in W for all z. For this proof we will assume that $\|Q_i(x)\| = 2$ for all i and all x.

There exists some polynomial q' such that S can contain at most $q'(n)$ of the strings in $\bigcup\{Q_i(<i, x, z>) : |z| = |x|\}$. There exists some polynomial q'' such that S can contain at most $q''(n)$ different *pairs* of strings in $\bigcup\{Q_i(<i, x, z>) : |z| = |x|\}$. We let $q(n) = q'(n) + q''(n) + 1$. It holds for all but finitely many x that $\|\{z{:}|z| = |x|\}\| > q(n)$ and $q''(|x|) > 2q'(|x|) + 1$. (Note that q'' is quadratic in q'.)

In the remainder of this proof we will consider only strings for which both inequalities hold. (We will use tabular lookup in the reduction for other strings.) Let Z consist of the lexicographically first $3 \times q(n)$ strings of length $|x|$. Only strings $<i, x, z>$ with $z \in Z$ may enter W. Let $U = \bigcup\{Q_i(<i, x, z>) : z \in Z\}$. Assume $U = \{u_1, \ldots, u_m\}$, where $u_i \in S \rightarrow u_{i+1} \in S$, and let $U' = \{u_{m-q'(n)+1}, \ldots, u_m\}$. Note that $U \cap S \subseteq U'$.

There are two cases.

Case 1. $\|U\| < q''(n)$.

As there are more than $3 \times q''(n)$ strings in Z it must hold that there are $z_1 \neq z_2 \neq z_3$ in Z such that $Q_i(<i, x, z_1>) = Q_i(<i, x, z_2>) = Q_i(<i, x, z_3>)$. Consider the truth table(s) produced on this input. By setting $<i, x, z>$ in

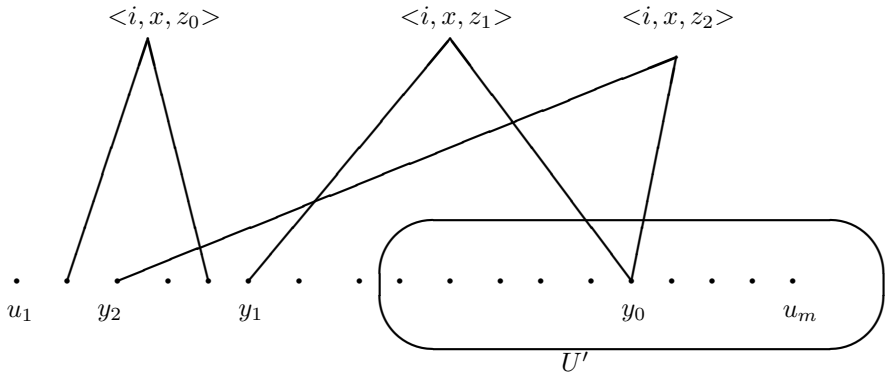


FIG. 3.1. *Possible query sets.*

or out of W we can eliminate at least half of the possible settings (element of A or not) of the two queries in $Q_i(<i, x, z_1>)$. As there are only four possible settings for these two strings, repeating this procedure for z_2 and z_3, we eliminate all possible settings, thereby forcing the fact that M_i is not a reduction from W to A.

Case 2. $\|U\| \geq q''(n)$. (It then follows that $\|U\| \geq 2 \times q'(n) + 1$.)

Either of the following holds (see also Figure 3.1):

Case 2a. There is a z in Z such that $Q_i(<i, x, z>) \cap U' = \emptyset$. Let the least such z be z_0. We let $<i, x, z_0>$ be in W iff $x \in K$ and let $<i, x, z> \notin W$ for all other z.

Case 2b: There is no such z. Then for each of the $\geq q'(n) + 1$ strings u_j in $U - U'$ there is a $z(u_j)$ in Z such that $\|Q_i(<i, x, z(u_j)>) \cap U'\| = \|Q_i(<i, x, z(u_j)>) \cap U - U'\| = 1$. It follows that there is a pair z_j, z'_j such that $\|Q_i(<i, x, z_j>) \cap Q_i(<i, x, z'_j>) \cap U'\| = 1$. Without loss of generality, we assume that z_1 and z_2 are minimal with this property and $z_1 < z_2$. Let

$$\{y_0\} = Q_i(<i, x, z_1>) \cap Q_i(<i, x, z_2>) \cap U',$$
$$\{y_1\} = Q_i(<i, x, z_1>) \cap U - U',$$
$$\{y_2\} = Q_i(<i, x, z_2>) \cap U - U'.$$

Then we know that $\{y_1, y_2\} \cap S = \emptyset$. In the following we will set, depending on the truth table computed by M_i on input $<i, x, z_1>$, the string $<i, x, z_1>$ in W such that $<i, x, z_2> \in W$ can be computed from the queries y_1 and y_2 and hence from queries to $A - S$.

We let $<i, x, z_2> \in W \leftrightarrow x \in K$. In the case we are considering here, only the strings $<i, x, z_1>$ and $<i, x, z_2>$ may enter W. So $<i, x, z> \notin W$ for all $z \in Z - \{z_1, z_2\}$.

Let α be the truth table computed by M_i on input $<i, x, z_1>$. There are sixteen possible truth tables that fall into four cases.

i. If $\alpha \equiv 1$, we let $<i, x, z_1> \notin W$ and if $\alpha \equiv 0$, we let $<i, x, z_1> \in W$. This prohibits M_i from being a 2-truth-table reduction of W to A. This covers two of the sixteen cases.

ii. If there is only *one* combination of $a, b \in \{0, 1\}$ for which $\alpha(a, b) = 1$, then $<i, x, z_1> \in W$, and if there is only *one* such combination where $\alpha(a, b) = 0$, then $<i, x, z_1> \notin W$. If this M_i is a 2-truth-

table reduction from W to A, then for this particular input we can compute $\chi_A(y_0)$ and $\chi_A(y_1)$ in polynomial time. This covers eight of the sixteen cases.

iii. If α is in fact a 1 truth table then the value of α depends on either y_0 or y_1. In the first case we put $<i, x, z_1>$ in W if and only if α takes on *true* if the entry for y_0 is fixed to *false*. Otherwise we let $<i, x, z_1> \in W$ iff $x \in K$.

iv. In the remaining two cases, α is either $\chi_A(y_0) = \chi_A(y_1)$ (equality) or $\chi_A(y_0) \oplus \chi_A(y_1)$ (parity). In the first case we let $<i, x, z_1> \in W$ and in the second case we let $<i, x, z_1> \notin W$ (then in both cases $y_0 \in A \Leftrightarrow y_1 \in A$.)

This completes the construction of W.

We will now show that we can use W to compute a $\leq^P_{2\text{-}tt}$-reduction for any set in *EXP* to $A - S$. Since W is exponential-time computable there is some $\leq^P_{2\text{-}tt}$-reduction of W to A. Let M_w be the machine that computes this reduction. For any set B in *EXP* let f_B be the many-one reduction of B to K. The $\leq^P_{2\text{-}tt}$-reduction from B to $A - S$ works as follows.

On input y, first compute $f_B(y)$. Let $x = f_B(y)$. Now use w and x to compute the set Z as above. Compute U, sort it according to the p-selector, find U', and find out which of the two cases (Case 2a or Case 2b) holds. (We have already argued that Case 1 cannot occur if w is the reduction from W to A.) In Case 2a, find the least z such that $Q_w(w, x, z) \cap U' = \emptyset$ and produce $Q_w(w, x, z)$ as a query set. In Case 2b find z_1 and z_2. We are in one of the subcases ii–iv. In subcases ii and iii we have either fixed the value of $\chi_A(y_0)$ or this value doesn't influence the value of the truth table, so we need only query y_1. Then we can compute membership of $x \in K$ from either the truth table produced by M_w on input $<i, x, z_1>$ or $<i, x, z_2>$, respectively. In subcase iv we fixed things such that $\chi_A(y_0) = \chi_A(y_1)$. We can query y_1 and y_2 and use these values as $\chi_A(y_0)$ and $\chi_A(y_2)$ in the truth table computed by M_w on input $<w, x, z_2>$. □

As every set in P is p-selective, we note the following corollary.

COROLLARY 3.14. *Theorems* 3.9, 3.10, *and* 3.13 *also hold when "p-selective" is replaced by "polynomial-time computable."*

We notice an interesting phenomenon here. In a recent paper, Buhrman, Fortnow, and Torenvliet [6] proved the existence of a 3-tt complete set in *EXP* that is not b-tt autoreducible. (Recall that a set A is r autoreducible if $A \leq^P_r A$ via a reduction that does not query its input.) Inspection of the proof shows that the set is constructed by diagonalizing against autoreductions on inputs in the set $\{0^{b(n)} : n \in \omega\}$, where $b(n)$ is some suitably chosen gap function. They prove that a 3-tt complete set A can be constructed such that every b-tt reduction (from A to A) that does not query its input must, for some n, incorrectly compute membership of $0^{b(n)}$ in A. Without essentially changing the proof, $b(n)$ can be chosen such that $\{0^{b(n)} : n \in \omega\}$ is a polynomial-time computable sparse set. The following corollary follows immediately from their proof.

COROLLARY 3.15 (see [6]). *There exists a* 3*-tt complete set* A *in EXP and a sparse set* S *in* P *such that* $A - S$ *is not b-tt hard for EXP.*

So Theorem 3.13 states an optimal result. A set that is not b-tt hard may of course still be Turing hard. Hence the corollary above hence does not rule out that every Turing-complete set may remain Turing complete when a sparse set is removed. However, we can show that such a result can only be obtained by nonrelativizing methods. We show that there exists an oracle set A relative to which *EXP* has a

$\leq_{tt}^{P}$-complete tally set T. Then $T - \{0\}^* = \emptyset$, which cannot be complete. We suspect the following to be a folk theorem. We prove it here just for completeness.

LEMMA 3.16. *If EXP $\subseteq$ P/poly, then there exists a tally set T that is $\leq_{tt}^{P}$-complete for EXP.*

Proof. It is well known that A is in $P/poly$ iff $A \leq_{tt}^{P} T$ for some tally set T. So from our hypothesis this gives us a tally set T, that is, truth-table *hard* for EXP. We will show how to construct a tally set T' (from T) that is complete for EXP.

The fact that there exists a tally set T that is hard for EXP gives us a truth-table reduction, say, by machine M_i from K to T. Now fix n and consider all strings of length n. Without loss of generality, we may assume that M_i queries on input x of length n, always the same strings to T, namely, $y_1, \ldots, y_{n^i}$, where $y_i = 0^i$. The idea is to find the minimal (in some way) setting of the y_j's in T such that $x \in K$ iff $M_i(x)$ accepts with this setting. Let x_j indicate the jth string of length n in lexicographic order. Let $P_j = \{<a_1, \ldots, a_{n^i}> : \text{when } a_i \text{ used as answer to query } y_i \text{ then } M_i(x_j) \text{ accepts iff } x_j \in K\}$.

P_j codes exactly those tally sets T' that, when used as an oracle for $M_i(x_j)$, let M_i compute the correct answer for $x_j \in K$. Note here that $\|P_j\| \leq 2^{n^i}$ for $0 \leq j < 2^n$. Set $P' = \bigcap_{i=1}^{2^n} P_i$ and let p_i be the ith projection of the tuple y, where y is the minimal $y \in P'$. Put $0^{<n,i>}$ in T' iff $p_i = 1$. Obviously, T' is tally and from the construction it is clear that T' is computable in exponential time. From the fact that T exists we get that T' exists and that $K \leq_{tt}^{P} T'$. □

THEOREM 3.17. *There exists an oracle A, such that there exists a $\leq_{tt}^{P^A}$-complete set B for EXP^A and a polynomial-time computable, sparse set S, such that $B - S$ is not $\leq_{tt}^{P^A}$-hard for EXP^A.*

Proof. Wilson [26] showed the existence of an oracle A where $EXP^A \subseteq P^A/poly$. Using this oracle, together with Lemma 3.16, we get that there exists a tally set T that is complete for EXP^A. Setting $B = T$ and $S = \{0\}^*$, we get that $B - S = \emptyset$ and $\emptyset$ is not $\leq_{tt}^{P^A}$-complete for EXP^A. □

4. Splittings of *EXP* complete sets. In this section we want to investigate to what extent one can *split* EXP-complete sets. A splitting of an r.e. (EXP) set is the construction of two r.e. (EXP) sets $A_0, A_1 \subseteq A$, such that $A_0 \bigcap A_1 = \emptyset$ and $A_0 \bigcup A_1 = A$. One of the questions to look at is: "Can this splitting be done so that both subsets have the same information as A?" For complete sets this would mean that the complete set can be split into subsets that are themselves again complete. This type of question has been studied in a recursion theoretical setting by Ladner [15]. He observed that there exist sets that are nonsplittable, or non-*mitotic*, as he called them. The recursion theoretical definition is as follows.

DEFINITION 4.1. *An r.e. set A is called mitotic iff there exist r.e. sets A_1 and A_0 such that*

1. $A_1 \subseteq A,\ A_0 \subseteq A,\ A_1 \bigcap A_0 = \emptyset,\ A_1 \bigcup A_0 = A.$
2. $A \equiv_T A_1 \equiv_T A_0.$

If additionally $A \equiv_m A_1 \equiv_m A_0$, then A is called m-mitotic.

Note here that point 2 in the definition can be weakened, in the case of $\leq_T$ reductions, to $A_1 \equiv_T A_0$. To see this note that $A \leq_T A_1 \oplus A_0$ and $A_1 \oplus A_0 \leq_T A_1$. To reduce A_1 to A, the reduction queries on input x whether or not x is in A. If this is not the case, it rejects straight out. Otherwise, it starts enumerating A_1 and A_0, since x must be in one of them.

Ambos-Spies [1] studied the complexity theoretical variant of mitotic sets and introduced the term *p-mitotic* sets. It is not clear how to define mitoticity in the complexity theoretical setting. Ambos-Spies introduced four definitions; two for the Turing reductions and two for the many-one reductions. One option is to change point 2 in Definition 4.1 into demanding $A \equiv_T^P A_1 \equiv_T^P A_0$. Since we are interested in complete sets for complexity classes, we could demand that A_0 and A_1 should be in the complexity class under consideration. A problem is that this definition cannot be weakened to $A_1 \equiv_T^P A_0$. (Take, for example, $\Sigma^* = A \bigcup \overline{A}$ for some *EXP* Turing-complete set A. Now both A and $\overline{A}$ are in *EXP*, split Σ^*, and are Turing equivalent but obviously do not Turing reduce to Σ^*.)

Ambos-Spies chose a Breidbart–Owings [4, 19] type of splitting (*by* another set).

DEFINITION 4.2 (see [1]). *A recursive set A is p-m(T) mitotic if there is a set $B \in P$ such that $A \equiv_{m(T)}^P A \bigcap B \equiv_{m(T)}^P A \bigcap \overline{B}$.*

When using this definition, the problem of reducing A_1 to A is settled for the Turing case. Namely, x is in A_1 iff x is in B and x is in A. A disadvantage of this definition, however, is that the requirement that the splitting has to be polynomial-time computable seems too strong. In order to capture this feeling, we also want to look at the definition discussed above. Note here also that since our main interest is in complete sets, we will not have the trouble that A_0 (or A_1) does not reduce to A. (This is because A is complete.)

DEFINITION 4.3. *An r.e. set A is called weakly p-T mitotic iff there exist r.e. sets A_1 and A_0, such that*

1. $A_1 \subseteq A,\ A_0 \subseteq A,\ \ A_1 \bigcap A_0 = \emptyset,\ \ A_1 \bigcup A_0 = A.$
2. $A \equiv_T^P A_1 \equiv_T^P A_0.$

If additionally $A \equiv_m^P A_1 \equiv_m^P A_0$, then A is called weakly p-m mitotic.

One of the questions that arise is: "Are $\leq_m^P$-complete sets for *EXP* (weakly) *p-m* mitotic?" In order to answer this question, we first take a look at the r.e.-complete sets. There it is known, due to Myhill [18], that all of the $\leq_m$-complete sets are isomorphic. Now, using the fact that K, the standard r.e. $\leq_m$-complete set, is m mitotic and that this property is preserved under isomorphisms, it follows that all $\leq_m$-complete sets are m mitotic. Unfortunately it is not known whether the $\leq_m^P$-complete sets for *EXP* are p isomorphic, but it is known that they are all 1-1 length increasing equivalent [3, 12, 25]. This is sufficient to prove that they are weakly *p-m* mitotic.

THEOREM 4.4. *All $\leq_m^P$-complete sets for EXP are weakly p-m mitotic.*

Proof. Let A be a $\leq_m^P$-complete set for *EXP*. Let $A \oplus A \leq_m^P A$ via f that is 1-1 and length increasing. Set $A_0 = \{y : \exists 0x[x \in A \wedge y = f(0x)]\}$. Since f is 1-1 and length increasing, A_0 is in *EXP*. It is also $\leq_m^P$-complete, because $x \in A$ iff $f(0x) \in A_0$. Now set $A_1 = A - A_0$. Then $A_0 \bigcup A_1 = A$ and $A_0 \bigcap A_1 = \emptyset$. It remains to show that A_1 is also $\leq_m^P$-complete. Let $A^1 = \{1x : x \in A\}$. Note that $A^1 \subseteq A \oplus A$ and is $\leq_m^P$-complete. Because f is 1-1, $1x \in A^1 \Rightarrow f(1x) \in A - A_0$, and $1x \notin A^1 \Rightarrow 1x \notin A \oplus A \Rightarrow f(1x) \notin A \Rightarrow f(1x) \notin A - A_0$. □

For *EXP*, the 1-1 length increasing property is enough to get weak *p-m* mitoticity. For *NEXP* the situation is somewhat different, because we do not know whether we have the length increasing property for complete sets. We do however have the 1-1 property and the fact that the reductions are not more than exponential length decreasing, i.e., $2^{|f(x)|} > |x|$. (The precise term here is "exponentially honest" [12].) The main problem, however, is that when applying the same proof as above, the set difference used to define $A_1 = A - A_0$ is not known to be in *NEXP*, because it is not known whether *NEXP* is closed under complementation (and thus under set difference). We *can* prove something that at first glance looks hopeful in order to prove weakly *p-m* mitoticity for *NEXP*-complete sets.

THEOREM 4.5. *Every $\leq_m^P$-complete set A for NEXP can be split into infinitely many disjoint subsets $A_1, A_2, \ldots$ such that $\bigcup_{i=0}^{\infty} A_i = A$, such that for all i, $A_i \subseteq A$ and A_i is complete for NEXP.*

Proof. We start the same way as in the *EXP* case. Let $A \oplus A \leq_m^P A$ via f that is 1-1 and exponentially honest. Set $A_0 = \{y : \exists\ 0x[y = f(0x) \wedge y \in A]\}$. Note that it is equivalent, in the definition of A_0, to say that $x \in A$ or $y \in A$, because f is a many-one reduction. Now A_0 is in *NEXP* : on input y guess $0x$ such that $f(0x) = y$. This can be done in nondeterministic exponential time because $|x| < 2^{|y|}$, by the exponential honesty of f. Now accept y iff $y \in A$. We define A_1 in a similar way. $A_1 = \{y : \exists\ 1x[y = f(1x) \wedge y \in A]\}$. We now have two complete sets A_0 and A_1 and some leftover of A, namely, $T_0 = A - (A_0 \bigcup A_1)$. At this point we repeat this procedure with A_0 resulting in A_{00} and A_{01} and again have some leftover T_1. The process of repeatedly splitting the set A_{0^ℓ} thus results in an infinite sequence of sets $(A_{0^\ell 1})_{\ell \in \omega}$ and a set $T = \bigcup_{i=0}^{\infty} T_i$ so that $(\bigcup_{\ell=0}^{\infty} A_{0^\ell 1}) \bigcup T = A$. Since T is countable (it is a subset of ω), we can add the ith element of T to $A_{0^i 1}$ resulting in a sequence $A'_{0^i 1}$ satisfying the properties of the theorem. □

Although this looks hopeful, the following example shows that the infinite version of mitoticity can be independent of mitoticity. Ladner [15] showed the existence of *nonmitotic* sets. Together with the following observation this yields the somewhat bizarre existence of a set that cannot be split into two parts but can be split into infinitely many parts of the same complexity.

OBSERVATION 4.6. *Every infinite r.e. set A can be split into infinitely many disjoint r.e. subsets $A_1, A_2, \ldots$ of A such that they remain in the same Turing degree as A.*

Proof. It is well known that every infinite r.e. set A has an infinite subset B that is recursive. Let B be such an infinite recursive subset of A. Since B is recursive and infinite, it is (recursively) isomorphic to Σ^*. So we can code A into B, i.e., let f be the isomorphism between Σ^* and B and define $A' = \{f(x) \mid x \in A\}$. Obviously A' is an infinite r.e. set and $A' \equiv_T (A - B)$. Furthermore there exists a T_1 such that $A = (A - B) \bigcup A' \bigcup T_1$. Now using the same "divide and split" trick as in the previous theorem, we get the desired sequence of subsets. □

We follow the same line as Ladner [15] and try to prove that there exist non-(weakly *p-m*) mitotic sets in *EXP*. We succeed in this and can also prove that those sets can be $\leq_3^P$-*tt*-complete. (Note that this also proves that the same result is true for *p-m* mitoticity.)

THEOREM 4.7. *There exists a set A in EXP that is not weakly p-m mitotic and $\leq_{3-tt}^P$-complete.*

Proof. In order to prove this, we prove the following: there exists a set A so that for all sets $A_0, A_1 \in EXP$ that split A, $A_0 \not\equiv_m^P A_1$. First note that if $A_i \notin EXP$, then $A_i \not\equiv_m^P A$ so, in the construction, we may assume that both A_0 and A_1 are recognized by exponential-time bounded machines. Or, equivalently, only pairs of exponential-time bounded machines represent candidate splittings of A against which we have to diagonalize.

Let $\{M_i\}_i$ be an enumeration of exponential-time machines that run in time 2^{n^i}, and let $\{f_k\}_k$ be an enumeration of polynomial-time many-one reductions computable in time n^k.

To construct A, we have requirements for all $n = \langle i, j, k, \ell \rangle$: if $L(M_i)$ and $L(M_j)$ split A, then either $L(M_j) \not\leq_m^P L(M_i)$ via f_ℓ or $L(M_i) \not\leq_m^P L(M_j)$ via f_k.

We introduce a function b having a set of strings to diagonalize on. Let $b(0) = 1$ and $b(i) = b(i-1)^{i-1} + 1$. At each stage n we will also define a number d_{n+1} such that if $n + 1 = \langle i', j', k', \ell' \rangle$, i.e., the *next* stage will diagonalize against ma-

chines $M_{i'}$ and $M_{j'}$ and functions $f_{k'}$ and $f_{\ell'}$, then d_{n+1} is some number satisfying $b(n+1)^{1/\max\{i',j'\}} > d_{n+1} > \max\{k',l'\} \times \log b(n+1)$. We will assume appropriate properties on the pairing function such that the interval from d_n to d_{n+1} is suitable for the construction. All strings in A with length between d_n and d_{n+1} will be defined at stage n.

The idea is to put three copies of K into A and make sure that at least one of the pairs $<i,x>$ $(i = 1,2,3)$ is in A. A is then $\leq^P_{3\text{-}tt}$(in fact $\leq^P_{3\text{-}d}$)-complete by the following reduction from K to A: $x \in K$ iff $<i,x>(i = 1,2,3)$ in A. On the other hand we can leave out at most two of the pairs $<i,x>$ to prevent A from becoming a mitotic set.

Construction:

stage 0: $A = \{<i,x> \mid i = 1,2,3 \text{ and } x \in K\}$. This is the "base set A." At any subsequent stage n of the construction, at most two strings y_0 and y_1, respectively, with $d_n \leq |y_0|, |y_1| < d_{n+1}$ and the string $0^{b(n)}$ will be added to or removed from A. A will be decided for all strings of length $\leq d_{n+1}$ after stage n.

Without loss of generality, we assume for this proof that the pairing function does not output strings that start with a 0. Then, we can define strings of the form $0^{b(i)}$ in or outside A without disturbing the completeness.

stage n:

Let $n = <i,j,k,\ell>$. This stage will diagonalize against the possibility that $A = L(M_i) \cup L(M_j)$ and/or that $L(M_i) \equiv^P_m L(M_j)$ via the functions f_k and f_ℓ Consider $f_k(0^{b(n)}) = y_0$ and $f_\ell(0^{b(n)}) = y_1$.

We have several cases to consider.

1. $y_0 = 0^{b(n)}$ or $y_1 = 0^{b(n)}$ in this case we put $0^{b(n)}$ into A.
2. $y_0 = y_1$, and not case 1. We have two subcases:
   (a) $|y_0| > d_n$. Leave y_0 out of A and put $0^{b(n)}$ in A.
   (b) $|y_0| \leq d_n$. Put $0^{b(n)}$ in A iff $y_0 \notin A$
3. $y_0 \neq y_1$, and not case 1. We have three subcases:
   (a) $|y_0| > d_n$ and $|y_1| > d_n$. Leave both y_0 and y_1 out of A and put $0^{b(n)}$ in A.
   (b) $|y_0| > d_n$ and $|y_1| \leq d_n$. ($|y_0| \leq d_n$ and $|y_1| > d_n$ is treated analogously.) We leave y_0 out of A and have two subcases:
      i. $y_1 \notin A$. Put $0^{b(n)}$ in A.
      ii. $y_1 \in A$. $0^{b(n)} \in A$ iff $M_j(y_1)$ accepts.
   (c) $|y_0| \leq d_n$ and $|y_1| \leq d_n$. We have three subcases:
      i. $y_1 \notin A$ and $y_0 \notin A$. Put $0^{b(n)}$ in A.
      ii. $y_1 \in A$ and $y_0 \notin A$. (The other way around is symmetric) $0^{b(n)} \in A$ iff $M_j(y_1)$ accepts.
      iii. $y_1 \in A$ and $y_0 \in A$. $0^{b(n)} \in A$ iff $M_i(y_0)$ accepts *and* $M_j(y_1)$ accepts.

end of stage n

End of Construction

This ends the construction of A. The proof of correctness of the construction is an analysis of the cases in the construction. First, A is exponential-time computable. A consists of strings $<i,x>$ where $x \in K$. Possible membership of these strings in A can be decided because K is exponential-time computable. To decide membership of other strings y in A, we need to compute $f_\ell(0^{b(n)})$ where $d_n \leq |y|$. This can be done in (linear) exponential time since $|y| \geq \max\{\ell,k\} \times \log b(n)$. Finally, to decide membership of $0^{b(n)}$ in A we sometimes need to compute $M_j(y)$ and/or $M_k(y)$ on some input y of length $\leq d_n$. This can be done in linear exponential time since $b(n) \geq d_n^{\max\{i,j\}}$.

Next, if A were m mitotic, then there exists a pair k, ℓ such that A is split into A_0 and A_1 and $A_0 \leq_m^P A_1$ via f_k and $A_1 \leq_m^P A_0$ via f_ℓ. Moreover, $A_0 \in EXP$ as is witnessed by machine M_i and $A_1 \in EXP$ as is witnessed by machine M_j.

The cases of the construction can be split into two types, namely, the type where both y_0 and y_1 are free, i.e., $> d_n$, and the type where either y_0 or y_1 is forced ($\leq d_n$). In the "free" cases we diagonalize by putting $0^{b(n)}$ in A and leaving both y_1 and y_0 out of A. Since $0^{b(n)} \in A$, it has to be in either A_0 or A_1, but if it is in either one then the corresponding reduction fails since its image is not in A (and therefore certainly not in the other part of A). In these cases we diagonalize directly against the many-one reductions. In the other cases we are forced to leave y_1 and y_0 in A in order not to destroy the work done at previous stages. But in these cases we are able to compute in exponential time the splittings we want to diagonalize against. We will show the correctness of case 3c(iii) in the construction. The other cases have a similar proof. In this case, both y_0 and y_1 are fixed and in A. By putting $0^{b(n)}$ in A, we force y_0 and y_1 for any *possible correct* splitting both to be in either A_0 or A_1. Yet, the machines M_i and M_j witness that y_0 and y_1 are in A_0 and A_1, respectively. On the other hand, by leaving $0^{b(n)}$ out of A, y_1 has to be in A_p and and y_0 in A_{1-p} ($p = 0, 1$). Since one of the machines rejects in this case, the other machine has to witness that y_0 and y_1 are both in A_0 or A_1. If, for instance, M_i witnesses that both strings are in A_0, then, since $0^{b(n)} \notin A_1$, reduction f_ℓ fails to be a reduction from A_1 to A_0. □

The next logical step would be to prove this result for Turing-complete sets and T mitoticity. This question remains open for further research.

Another line of splittings in recursion theory is the existence of a splitting of an r.e. set A in A_0 and A_1 that are incomparable. Examples of this are the splitting theorem of Sacks [20] and the time bounded versions by Ladner [16]. The next theorem is in a way a counterpart to this.

In the original splittings one gets the following structure: A_0 and A_1 are Turing (or many one) incomparable but do reduce to A, thus achieving that A does not reduce to A_0 or A_1, i.e., A_0 and A_1 are strictly below A. In the next theorem the sets A_0 and A_1 are strictly below A, but are in the same many-one degree. Seen in another light, this theorem can be seen as a generalization of the fact that there exists $\leq_{2-d}^P$-complete sets for EXP that are not $\leq_m^P$-complete [25, 12].

THEOREM 4.8. *If A is $\leq_m^P$-complete for EXP, then A can be split into A_0 and A_1, such that*

- $A_0 \equiv_m^P A_1$.
- *A_0 and A_1 are $\leq_{2\text{-}d}^P$-complete for EXP but not $\leq_m^P$-complete.*

Proof. Let A be $\leq_m^P$-complete and K be the standard $\leq_m^P$-complete set. Since the $\leq_m^P$-complete sets for EXP are 1-1 length increasing equivalent, we can construct the following length increasing 1-1 function h from A to A. Let f be the 1-1, length increasing reduction from A to K, and let g be the one from K to A. Let $h(x) = f(g(x))$. We say that x is a *root* if $h^{-1}(x)$ is undefined and x is *on a chain* if $h^{-1}(x)$ is defined.

One possible way to construct A_0 and A_1 is as follows (the real construction follows later): $A_0 = \{x \mid x \in A \text{ and } x \text{ is a root } \} \bigcup \{x \mid x \in A \text{ and } x \text{ is on a chain and } h^i(x_r) = x \text{ and } x_r \text{ is a root and } i \text{ is even}\}$ and $A_1 = \{x \mid x \in A \text{ and } x \text{ is on a chain and } h^i(x_r) = x \text{ and } x_r \text{ is a root and } i \text{ is odd}\}$.

Clearly A_0 and A_1 split A, are in EXP, and $A_0 \equiv_m^P A_1$ via h. A_0 and A_1 are $\leq_{2\text{-}d}^P$-complete: $x \in A \iff (x \in A_0 \text{ or } x \in A_1) \iff (x \in A_0 \text{ or } h(x) \in A_0)$. The only thing to do now is to show that A_0 and A_1 are not $\leq_m^P$-complete. Note here that in order to get the above properties (i.e., splitting of A, the $\equiv_m^P$, and the $\leq_{2\text{-}d}^P$-completeness)

it doesn't matter if the roots are in A_0 or A_1 as long as it holds that $x \in A_i$, then $h(x) \in A_{1-i}$. This gives enough freedom to diagonalize against $\leq_m^P$-reductions. We are going to construct a set $W \in EXP$ so that $W \not\leq_m^P A_0$. Note that then A_1 cannot be $\leq_m^P$-complete either. Again let $\{f_i\}_i$ be an enumeration of $\leq_m^P$-reductions such that f_i runs in time $n^i + i$. We also need a function $b(n)$ to denote the set of strings to diagonalize on. Let $b(0) = 1$ and $b(i+1) = b(i)^i + 1$. W is going to be a subset of 0^*.

We construct W, A_0, and A_1 in stages such that elements of A_0 and A_1 are either roots or successive elements on a chain and W is exponential-time computable, but reducible to neither A_0 nor A_1.

At stage 0, $W = A_0 = A_1 = \emptyset$.

stage n:

We have constructed W, A_0, and A_1 up to strings of length $\leq b(n-1)^{n-1}$. We simulate $f_n(0^{b(n)}) = y$. We now have three cases for the construction of W:

1. $|y| \leq b(n-1)^{n-1}$. Put $0^{b(n)}$ in W iff $y \notin A_0$.
2. $\exists\, y' \leq b(n-1)^{n-1}$ such that $h^i(y') = y$. Put $0^{b(n)}$ in W iff $y' \notin A_{i \bmod 2}$.
3. Otherwise put $0^{b(n)}$ in W.

This ends the construction of W.

Construction of A_0 and A_1.

Let $b(n-1)^{n-1} < |x| < b(n)^n + 1$ and $x \in A$.

1. $x \neq y$.
   (a) x is a root.
   - $\exists\, i$ such that $h^i(x) = y$
     - put x in A_0 iff i is odd.
     - put x in A_1 iff i is even.
   - put x in A_0.

   (b) x is on a chain, x_r is the root of x, $x_r \in A_j$, and $h^i(x_r) = x$. Put x in $A_{(i+j) \bmod 2}$.
2. $x = y$. Put x in A_0 iff $0^{b(n)} \notin W$.

end of stage n

It is not hard to see that W, A_0, and A_1 are all in EXP. Furthermore A_0 and A_1 split A and $A_0 \equiv_m^P A_1$. It remains to be shown that $W \not\leq_m^P A_0$. Suppose it is via reduction f_j; then $f_j(0^{b(j)}) = y$. For cases 1 and 2 in the above construction, $0^{b(j)} \in W$ iff $y \notin A_0$, and from the construction of A_0 and case 3, $0^{b(j)} \in W$ and $y \notin A_0$. □

Acknowledgments. We thank Peter van Emde Boas for discussions about the results. We thank Steven Homer, Dick de Jongh, and Jan van Neerven for discussions and ideas. We thank Stephen Wilcox for pointing out an error that we thought was grammatical but that turned out to be mathematical.

REFERENCES

[1] K. Ambos-spies, *On the structure of polynomial time degrees*, in Proc. Symposium on Theorectical Aspects of Computer Science, Lecture Notes in Comput. Sci. 166, M. Fontet and K. Mehlhorn, eds., Springer-Verlag, Berlin, 1984, pp. 198–208.

[2] J. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity* I, Springer-Verlag, Berlin, 1988.

[3] L. Berman, *Polynomial Reducibilities and Complete Sets*, Ph.D. thesis, Cornell University, Ithaca, NY, 1977.

[4] S. Breidbart, *On splitting recursive sets*, J. Comput. System Sci., 17 (1978), pp. 56–64.

[5] H. Buhrman, *Resource Bounded Reductions*, Ph.D. thesis, University of Amsterdam, 1993.

[6] H. Buhrman, L. Fortnow, and L. Torenvliet, *Using autoreducibility in separating complexity classes*, in Proc. 36th Annual IEEE Symposium on Foundations of Computer Science, Milwaukee, WI, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 520–527.

[7] H. Buhrman, S. Homer, and L. Torenvliet, *On complete sets for nondeterministic classes*, Math. Systems Theory, 24 (1991), pp. 179–200.
[8] H. Buhrman, E. Spaan, and L. Torenvliet, *The relative power of logspace and polynomial time reductions*, Comput. Complexity, 3 (1993), pp. 231–244.
[9] H. Buhrman and L. Torenvliet, *On the structure of complete sets*, in Proc. 9th Annual IEEE Conference on Structure in Complexity Theory, Amsterdam, Holland, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 118–133.
[10] H. Buhrman, L. Torenvliet, and P. V. Emde Boas, *Twenty questions to a p-selector*, Inform. Process. Lett., 48 (1993), pp. 201–204.
[11] H. Buhrman, and L. Torenvliet, *P-selective self-reducible sets: A new characterization of P*, J. Comput. System Sci., 53 (1996), pp. 210–217.
[12] K. Ganesan and S. Homer, *Complete problems and strong polynomial reducibilities*, SIAM J. Comput., 21 (1992), pp. 733–742.
[13] C. Jockusch, *Semirecursive sets and positive reducibility*, Trans. Amer. Math. Soc., 131 (1968), pp. 420–436.
[14] J. Kelly, *General Topology*, D. van Nostrand Company Inc., New York, 1955.
[15] R. Ladner, *Mitotic recursively enumerable sets*, J. Symbolic Logic, 38 (1973), pp. 199–211.
[16] R. Ladner, *On the structure of polynomial time reducibility*, J. Assoc. Comput. Mach., 22 (1975), pp. 155–171.
[17] R. Ladner, N. Lynch, and A. Selman, *A comparison of polynomial time reducibilities*, Theoret. Comput. Sci., 1 (1975), pp. 103–123.
[18] J. Myhill, *Creative sets,* Zeit. Math. Log. Grund. Math., 1 (1955), pp. 97–108.
[19] J. Owings, *Splitting a context-sensitive set*, J. Comput. System Sci., 10 (1975), pp. 83–87.
[20] G. Sacks, *On degrees less than* $\mathbf{0}'$, Ann. of Math., 2 (1963), pp. 211–231.
[21] U. Schöning, *Complete sets and closeness to complexity classes*, Math. Systems Theory, 19 (1986), pp. 29–41.
[22] A. Selman, *P-selective sets, tally languages, and the behavior of polynomial time reducibilities on NP*, Math. Systems Theory, 13 (1979), pp. 55–65.
[23] S. Tang, B. Fu, and T. Liu, *Exponential time and subexponential time sets*, Theoret. Comput. Sci., 115 (1993), pp. 371–381.
[24] S. Toda, *On polynomial-time truth-table reducibilities of intractable sets to P-selective sets*, Math. Systems Theory, 24 (1991), pp. 69–82.
[25] O. Watanabe, *A comparison of polynomial time completeness notions*, Theoret. Comput. Sci., 54 (1987), pp. 249–265.
[26] C. Wilson, *Relativized circuit complexity*, J. Comput. System Sci., 31 (1985), pp. 169–181.

SIAM J. COMPUT.
Vol. 27, No. 3, pp. 654–667, June 1998
© 1998 Society for Industrial and Applied Mathematics
004

CONSTRUCTING LEVELS IN ARRANGEMENTS AND HIGHER ORDER VORONOI DIAGRAMS *

PANKAJ K. AGARWAL†, MARK DE BERG‡, JIŘÍ MATOUŠEK§, AND OTFRIED SCHWARZKOPF¶

Abstract. We give simple randomized incremental algorithms for computing the $\leq k$-level in an arrangement of n lines in the plane or in an arrangement of n planes in $\mathbb{R}^3$. The expected running time of our algorithms is $O(nk + n\alpha(n)\log n)$ for the planar case and $O(nk^2 + n\log^3 n)$ for the three-dimensional case. Both bounds are optimal unless k is very small. The algorithm generalizes to computing the $\leq k$-level in an arrangement of discs or x-monotone Jordan curves in the plane. Our approach can also compute the k-level; this yields a randomized algorithm for computing the order-k Voronoi diagram of n points in the plane in expected time $O(k(n-k)\log n + n\log^3 n)$.

Key words. arrangements, random sampling, Voronoi diagrams

AMS subject classifications. 65Y25, 68Q25, 68U05

PII. S0097539795281840

1. Introduction.

Arrangements of hyperplanes have been studied for a long time in combinatorial and computational geometry and yet they have kept some of their secrets. Some of the intriguing open questions are related to the concept of *levels*. We say that a point p is at level k with respect to a set H of nonvertical hyperplanes in $\mathbb{R}^d$ if there are exactly k hyperplanes in H that lie strictly above p. The k-level of an arrangement $\mathcal{A}(H)$ of hyperplanes is the closure of all $(d-1)$-cells of $\mathcal{A}(H)$ whose interior points have level k with respect to H. It is a monotone piecewise-linear surface; see Figure 1. The $\leq k$-level of $\mathcal{A}(H)$ is the complex induced by all cells of $\mathcal{A}(H)$ lying on or above the k-level. The k-level and the $\leq k$-level of arrangements of monotone surfaces are defined analogously. In fact, one can give a more general definition of levels, which is useful in certain applications. Given a family Γ of subsets (also called *ranges*) of $\mathbb{R}^d$, define the level of a point p with respect to Γ to be the number of ranges that contain p in their interior; k-level and $\leq k$-level are defined as above. For a set H of hyperplanes, if we choose ranges to be the halfspaces lying below the hyperplanes of H, then the level of p is the same under the two definitions.

*Received by the editors February 22, 1995; accepted for publication (in revised form) March 11, 1996. The work of the first author was supported by National Science Foundation grant CCR-93-01259 and by an NYI award. The second and fourth authors were supported by the Netherlands' Organization for Scientific Research (NWO) and partially supported by ESPRIT Basic Research Action 7141 (project ALCOM II: Algorithms and Complexity). Work by the third author was supported by Charles University grant 351, Czech Republic grant GAČR 201/93/2167, and EC Cooperative Action IC-1000 (project ALTEC: Algorithms for Future Technologies). Part of this research was done when the first and fourth authors visited Charles University and when the first author visited Utrecht University. These visits were supported by Charles University and NWO.

http://www.siam.org/journals/sicomp/27-3/28184.html

†Department of Computer Science, Box 90129, Duke University, Durham, NC 27708-0129 (pankaj@cs.duke.edu).

‡Vakgroep Informatica, Universiteit Utrecht, Postbus 80.089, 3508 TB Utrecht, the Netherlands (markdb@cs.ruu.nl).

§Department of Applied Mathematics, Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic (matousek@kam.mff.cuni.cz).

¶Department of Computer Science, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, the Netherlands. Current address: Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong (otfried@cs.ust.hk).

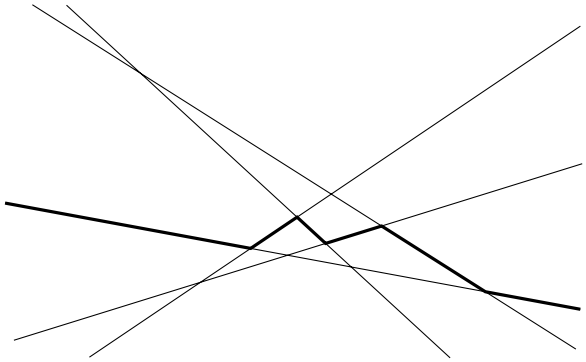


FIG. 1. *The* 2*-level in an arrangement of lines.*

We will not distinguish between the two definitions of levels, as it will be clear from the context to which of them we are referring.

The maximum combinatorial complexity of the $\leq k$-level in an arrangement of hyperplanes in $\mathbb{R}^d$ is known precisely. Using a probabilistic argument, Clarkson and Shor [CS89] proved that it is $\Theta(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$. By a similar technique, Sharir [Sha91] proved that the maximum complexity of the $\leq k$-level for a set of n discs is $\Theta(nk)$. He also proved that the complexity of the $\leq k$-level for n x-monotone Jordan curves, each pair of which intersects in at most s points, is $O(k^2 \lambda_s(n/k))$. Here $\lambda_s(m)$ is the maximum length of an (m, s) Davenport–Schinzel sequence; $\lambda_s(m)$ is roughly linear in m for any constant s [ASS89].

The problem of efficiently computing the $\leq k$-level has not been resolved completely. Mulmuley [Mul91b] gave a randomized incremental algorithm for computing the $\leq k$-level in hyperplane arrangements in any dimension. For $d \geq 4$, the expected running time of his algorithm is $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$, which is optimal. For $d = 2, 3$ the expected running time of his algorithm is $O(nk^{\lceil d/2 \rceil} \log(n/k))$, which exceeds the output size by a logarithmic factor. Recently, Everett, Robert, and van Kreveld [ERvK96] gave an optimal $O(n \log n + nk)$ expected time randomized algorithm for computing the $\leq k$-level in an arrangement of lines in the plane. Their algorithm does not easily extend to more general ranges.

Mulmuley's algorithm can be applied to compute the $\leq k$-level of arrangements of x-monotone Jordan curves in the plane, but it is not clear how to generalize it for computing the $\leq k$-level for more general ranges like discs. Sharir [Sha91] presented a divide-and-conquer algorithm for computing the $\leq k$-level of rather general ranges in the plane. Its worst-case running time is roughly $\log^2 n$ times the maximum size of the $\leq k$-level.

In this paper we give a somewhat different randomized incremental algorithm for computing the $\leq k$-level whose expected running time is $O(nk + n \log n\, \alpha(n))$ in the plane and $O(nk^2 + n \log^3 n)$ in 3-space, which is worst-case optimal unless k is very small. The main difference of this algorithm compared with Mulmuley's algorithm [Mul91b, Mul93] is as follows. Mulmuley's algorithm inserts the hyperplanes one by one in a random order and maintains the $\leq k$-level of $\mathcal{A}(R)$, the arrangement of the hyperplanes that have already been inserted. When adding a new hyperplane h, the algorithm first updates the arrangement locally at cells that are intersected by h. Then it removes cells that have "fallen off" because they are now on level $k+1$; Mulmuley calls this the *peeling step*. Our algorithm maintains a part of $\mathcal{A}(R)$ that is in general

smaller than the $\leq k$-level. Namely, when inserting a new hyperplane, the algorithm estimates the level of each of the newly created cells of the current cell complex with respect to H, the full set of hyperplanes. Mulmuley's algorithm, in contrast, looks at the level with respect to R, the set of hyperplanes already inserted. As soon as our estimate shows that a cell lies completely outside the $\leq k$-level, it is discarded. This strategy can also be used in situations where Mulmuley's algorithm has difficulty accessing the cells of the current complex that must be peeled off. This is, for instance, the case where one wants to compute the $\leq k$-level in an arrangement of discs.

Our approach can also be used to compute the k-level in an arrangement. This time a cell of the current cell complex is discarded as soon as it becomes clear that it does not intersect the k-level of $\mathcal{A}(H)$. The complexity of this algorithm depends on combinatorial bounds on the complexity of a k-level, and here the knowledge is less satisfactory than for the $\leq k$-level. Lovász proved that a k-level in an arrangement of n lines in the plane has $O(n\sqrt{k})$ vertices [Lov71], and Erdős et al. [ELSS73] proved a lower bound of $\Omega(n \log(k+1))$. There was no progress on this problem for the last twenty-five years except for a slightly improved upper bound by Pach, Steiger, and Szemerédi [PSS92]. Recently, Dey [Dey97] improved the upper bound to $O(nk^{1/3})$. See [AAS97, DE93, ZV92] for known results on k-levels in higher dimensions. Edelsbrunner and Welzl [EW86] gave an algorithm for computing the k-level in the plane, which was later slightly improved by Cole, Sharir, and Yap [CSY87] to $O(n \log n + m \log^2 k)$ running time, where m stands for the actual complexity of the k-level. Our algorithm yields $O(nk^{1/3} \log^{2/3} n + n \log^2 n)$ expected time complexity. (If one could prove better worst-case bounds for the k-level maximum complexity, we would get an improvement in the algorithm's complexity as well; see section 4 for exact bounds.) This compares favorably to the worst-case running times of previous algorithms; however, our algorithm is not fully output sensitive (its running time depends on the complexity of levels in arrangements of various random subsets of the given lines). In principle, our algorithm for computing the k-level also works in higher dimensions, and its running time can be obtained by plugging the known bounds on the complexity of the k-level in higher dimensions [ABFK92, DE93, ZV92] into the analysis of our algorithm. We, however, do not discuss it in this paper, because our main interest lies in the special three-dimensional case discussed next.

If all the hyperplanes are tangent to the unit paraboloid, the maximum complexity of the k-level of a three-dimensional arrangement is known to be $\Theta(k(n-k))$. This situation arises when the planes are the images of a set of points in the plane under the transformation that maps the order-k Voronoi diagram of these points to the k-level of the planes (see, e.g., [Ede87]). Most known algorithms for computing the k-level in three-dimensional space actually compute the $\leq k$-level [Mul91b, CE87, BDT93]. Since the complexity of the $\leq k$-level is $\Theta(nk^2)$ in the situation sketched above, the running time of these algorithms is at least $\Omega(nk^2 + n \log n)$. The randomized incremental algorithm by Aurenhammer and Schwarzkopf [AS92] maintains only the k-level, but it can be shown that any randomized incremental algorithm that maintains the k-level of the intermediate arrangements must take time $\Omega(nk^2)$ as well, since the expected number of structural changes in the k-level is $\Omega(nk^2)$ [AS92]. The only algorithm that approaches the desired $O(k(n-k))$ time bound was presented by Clarkson [Cla87], and it runs in time $O(n^{1+\varepsilon}k)$, where $\varepsilon > 0$ is an arbitrarily small constant. His algorithm can probably be improved somewhat by using more recent results on geometric cuttings.

We show that our algorithm for computing the k-level for n planes tangent to the unit paraboloid runs in $O(k(n-k)\log n + n\log^3 n)$ expected time in this case. We conjecture that the running time is in fact $O(k(n-k)+n\log n)$, which is asymptotically optimal, but currently we cannot prove it.

2. Preliminaries.

Arrangements. Let H be a set of n hyperplanes in d-dimensional space. We denote the arrangement of H by $\mathcal{A}(H)$. We regard it as a cell complex with cells (also called faces) of dimensions 0 to d that are relatively open. We define the *level* of a cell of $\mathcal{A}(H)$ as the level of any point of its relative interior.

For simplicity of exposition, we assume that the hyperplanes are in general position. This assumption can be removed by a more careful (and technically a little more complicated) treatment or by standard perturbation arguments [Ede87].

Canonical triangulations. Let $\mathcal{C}$ be a subcomplex of $\mathcal{A}(H)$, that is, a collection of cells of $\mathcal{A}(H)$ such that whenever $C \in \mathcal{C}$ and C' is a face of C, then also $C' \in \mathcal{C}$. The *canonical triangulation*[1] of $\mathcal{C}$ ([Cla88]), which we denote by $\mathcal{C}^\nabla$, is defined as follows. Let C be a j-dimensional cell of $\mathcal{C}$ (thus, a convex polytope), and let v be the bottom vertex of C, that is, the lexicographically smallest vertex of C. If $j = 1$, then C is a segment and it is already a (one-dimensional) simplex. If $j > 1$, then we recursively triangulate the $(j-1)$-dimensional faces of C and extend each $(j-1)$-simplex to a j-simplex using the vertex v. (Unbounded cells require some care in this definition [Cla88].) The canonical triangulation of a cell with m vertices has $O(m)$ simplices. The canonical triangulation $\mathcal{C}^\nabla$ of the subcomplex $\mathcal{C}$ is the simplicial complex obtained by triangulating each cell of $\mathcal{C}$ in this manner.

Let R be a subset of H. For a (relatively open) simplex $\Delta \in \mathcal{A}(R)^\nabla$, let $K(\Delta)$ denote the set of hyperplanes of H intersecting Δ, and put $w(\Delta) = |K(\Delta)| + 1$. The hyperplanes from $K(\Delta)$ are said to be *in conflict with* Δ, and $w(\Delta)$ is called the *weight* of Δ.

For each simplex Δ of the canonical triangulation, let $D(\Delta) \subseteq H$ be a set of hyperplanes such that Δ appears in the canonical triangulation of the arrangement of $D(\Delta)$, but does not appear in the canonical triangulation of $\mathcal{A}(D')$ for any proper subset $D' \subset D(\Delta)$. It can be shown that if H is in general position, $D(\Delta)$ is unique and its size is bounded by a constant $b = b(d)$. This $D(\Delta)$ is called the *defining set* of Δ. For instance, for $d = 2$ a typical triangle Δ is defined by five lines—the two lines whose intersection is the bottom vertex, the line containing the side opposite to the bottom vertex, and the two lines intersecting this side at the other two vertices of Δ, so $b(2) = 5$. The following fact is often used in the analysis of randomized geometric algorithms; among others, it captures the above-mentioned uniqueness of the defining set. For a proof and detailed discussion, see Chazelle and Friedman [CF90, sec. 5].

FACT 2.1. *Let $S \subseteq H$, and let Δ be a simplex of $\mathcal{A}(S)^\nabla$. Then the following condition holds: for any $R \subseteq H$, $\Delta \in \mathcal{A}(R)^\nabla$ if and only if $D(\Delta) \subseteq R$ and $R \cap K(\Delta) = \emptyset$.*

When generalizing the algorithm below to computing the $\le k$-level in more general arrangements (for example, in an arrangement of discs), we need to define an appropriate analog of the canonical triangulation having the above property.

A tool for analyzing randomized incremental algorithms. We now discuss a random-sampling result that we need for the analysis of our algorithms. We do not state it in a full generality but only in the specific setting in which we need it.

[1] Sometimes the name *bottom vertex triangulation* is used in the literature.

LEMMA 2.2. *For each $R \subseteq H$, let $\mathcal{C}(R)$ be a subcomplex of the arrangement $\mathcal{A}(R)$. Assume that the subcomplexes have the following "monotonicity" property:*

(∗) *Let $R' \subseteq R \subseteq H$, let C be a cell of $\mathcal{C}(R)$, and let C' be the cell of $\mathcal{A}(R')$ containing C. Then $C' \in \mathcal{C}(R')$.*

(For instance, $\mathcal{C}(R)$ may be the collection of all cells in $\mathcal{A}(R)$ intersecting some fixed set $X \subseteq \mathbb{R}^d$, plus their lower-dimensional faces. Then (∗) clearly holds. In our applications, the definition of $\mathcal{C}(R)$ will be a bit more technical, but it will have a similar flavor.)

Let $f : \mathbb{N} \to \mathbb{R}^+$ be a nondecreasing function so that $f(r)$ is an upper bound for the expected total number of vertices of $\mathcal{C}(R)$, where R is a randomly chosen r-element subset of H. Then we have the following:

(i) *Let R be a randomly chosen r-element subset of H, and let c be a constant. Then*

$$\mathbf{E}\left[\sum_{\Delta\in\mathcal{C}(R)^\nabla} w(\Delta)^c\right] = O\left(\left(\frac{n}{r}\right)^c f(r)\right).$$

(ii) *Consider a randomized incremental algorithm that inserts the hyperplanes of H one by one in a random order and maintains $\mathcal{C}(R)^\nabla$, where $R = \{h_1, h_2, \ldots, h_r\}$ is the set of hyperplanes already inserted. Let $\mathcal{N}_r$ denote the set of simplices newly created at step r, that is, $\mathcal{N}_r = \mathcal{C}(\{h_1, h_2, \ldots, h_r\})^\nabla \setminus \mathcal{C}(\{h_1, h_2, \ldots, h_{r-1}\})^\nabla$. Then for any constant $c \geq 0$ we have*

$$\mathbf{E}\left[\sum_{\Delta\in\mathcal{N}_r} w(\Delta)^c\right] = O\left(\frac{n^c}{r^{c+1}} f(r)\right).$$

Part (i) says that, under condition (∗), the expected average weight $w(\Delta)$ of a simplex in $\mathcal{C}(R)^\nabla$ is about n/r even if we take the cth degree averages. Often the amount of work that a randomized algorithm performs at step r is directly related to the weight of the newly created simplices; part (ii) can then be used to analyze the running time.

Sketch of proof. Results of this type were first obtained by Chazelle et al. [CEG+93], who essentially proved (ii) with $c = 1$ (in a more general setting) by analyzing a randomized incremental algorithm. The general case, where $c > 1$, can be obtained by an extension of their analysis, as is shown by de Berg, Dobrindt, and Schwarzkopf [dBDS94]. A somewhat different approach, going via the "static" part (i) and generalizing an approach of Chazelle and Friedman [CF90], is presented by Agarwal, Matoušek, and Schwarzkopf [AMS97]. In order to apply the general framework of [AMS97] to our situation, we need the following properties of the canonical triangulation of $\mathcal{C}(R)$:

(a) If a simplex Δ is present in $\mathcal{C}(R)^\nabla$, then $D(\Delta) \subseteq R$ and $K(\Delta) \cap R = \emptyset$.

(b) For any $\Delta \in \mathcal{C}(R)^\nabla$, we have $\Delta \in \mathcal{C}(R')^\nabla$ whenever $R' \subseteq R$ and $D(\Delta) \subseteq R'$.

Condition (a) is immediate from Fact 2.1, and condition (b) follows from Fact 2.1 together with property (∗). Under these conditions, (i) is proved by Agarwal, Matoušek, and Schwarzkopf [AMS97]. Part (ii) can now be derived from (i) by *backward analysis*: the simplices of $\mathcal{N}_r$ are those destroyed by deleting the hyperplane h_r from $R = \{h_1, h_2, \ldots, h_r\}$. Since the order of hyperplanes is random, h_r is a random hyperplane of R. Each simplex of $\mathcal{C}(R)^\nabla$ is only destroyed by deleting h_r if $h_r \in D(\Delta)$.

Therefore, the expected sum $\sum_\Delta w(\Delta)^c$ over the simplices destroyed by deleting a random hyperplane of R is at most b/r times the expectation of $\sum_{\Delta\in\mathcal{C}(R)^\nabla} w(\Delta)^c$, and since R is a random r-element subset of H, we can use (i). □

3. Computing the $\leq k$-level and k-level for hyperplanes.

Outline of the algorithm. We describe the algorithm for an arbitrary dimension $d \geq 2$ because it may be a useful alternative to Mulmuley's algorithm even for $d \geq 4$ (although the asymptotic running time for $d \geq 4$ is the same as that of his algorithm).

We first generate a random permutation $h_1, h_2, \ldots, h_n$ of H and then insert the hyperplanes one by one in this order. Let R denote the set $\{h_1, \ldots, h_r\}$ of hyperplanes inserted in the first r steps. As the hyperplanes are inserted, the algorithm maintains the following structures:

- The canonical triangulation $\mathcal{K}_r^\nabla$ of a subcomplex $\mathcal{K}_r$ of $\mathcal{A}(R)$ (more precisely, we store the simplices as well as their adjacency relations).
- The conflict lists: for every simplex $\Delta \in \mathcal{K}_r^\nabla$, the list $K(\Delta)$ of hyperplanes of $H \setminus R$ intersecting its relative interior, and for every hyperplane $h \in H \setminus R$, the list of all simplices with $h \in K(\Delta)$.
- For every simplex $\Delta \in \mathcal{K}_r^\nabla$, the level ℓ_Δ of one of its (arbitrarily chosen) interior points with respect to $\mathcal{A}(H)$.

At a high level, the insertion of a new hyperplane h_r can be described as follows.

1. Find all cells of $\mathcal{K}_{r-1}$ intersected by h_r using the conflict lists. Split such cells, retriangulate them as necessary, update the conflict lists, and compute the level information for the new simplices.
2. Test each of the newly created cells to see whether it is *active* (the meaning of this will be described below). If a new cell is not active, its simplices and their conflict lists are discarded. The resulting complex is $\mathcal{K}_r$.

We now discuss these steps in more detail.

Updating the information. In the first step, we identify all the simplices in $\mathcal{K}_{r-1}^\nabla$ intersected by h_r using the conflict lists. Since we know the adjacency relations among the simplices, we can compute the intersected simplices as a collection of groups, one for each cell that h_r intersects. Consider such a group, and let C be the corresponding cell. All the simplices in the group have to be deleted and replaced by a number of new simplices. We first consider the situation for $d = 2$, which is illustrated in Figure 2. To deal with the part of C that lies on the same side of h_r as the bottom vertex v of C, we draw new diagonals from v to the two intersection points of h_r with the boundary of C; this creates three new simplices. The part of C lying on the opposite side of h_r is simply triangulated from scratch. This way the canonical triangulations of the two cells that result from the splitting of C are constructed in time that is linear in the number of new simplices that are created. The adjacency relations among the simplices can easily be determined during the process.

Things are not very different in higher dimensions. Consider a three-dimensional cell C. We first retriangulate the two-dimensional facets that are intersected, in the way we just described. We also triangulate the facet $C \cap h_r$. It remains to connect the triangles of the boundary triangulation to the correct bottom vertex (this is necessary only if they were not already connected to this vertex in the triangulation of C, of course); this is analogous to the two-dimensional case. In general, in dimension d we first treat the parts of the $(d-1)$-facets incident to intersected simplices recursively, and then we connect the $(d-1)$-simplices that still need to be extended to d-simplices to the correct bottom vertex.

Our second task is to compute the conflict lists of the new simplices. Let C be a cell that has been split by h_r into two new cells C' and C'', where C' is the cell that

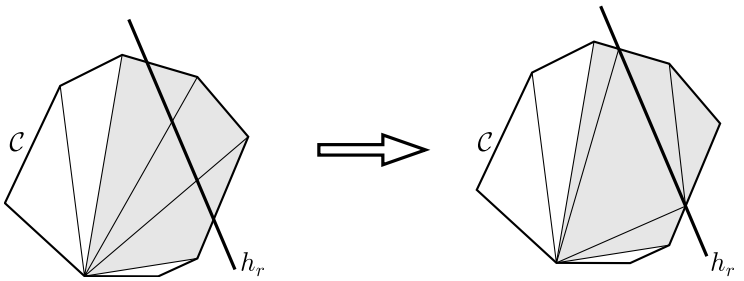


FIG. 2. *Retriangulation of a cell that is split.*

contains the bottom vertex of C. Let S be the set of new simplices in C' and C''. Some of the simplices of C' may already have existed in C; these simplices already have the correct conflict list and are not present in S. The union of the conflict lists for the simplices in S is the same as the union of the conflict lists of the simplices of C that were destroyed by h_r (minus h_r itself). We denote this set of hyperplanes by $K(S)$. To find the conflict lists for the simplices in S, we first determine for each hyperplane $h \in K(S)$ one simplex of S that it intersects — its *initial simplex* — as follows.

Consider a hyperplane $h \in K(S)$. If h intersects some simplex of S in an edge that is also an edge of the old cell C, then this simplex can serve as initial simplex for h. We can find it through the conflict list of a destroyed simplex of C that contained that edge. If h intersects none of the simplices of S in an edge of C, then h must separate the bottom vertex of C' from $C \cap h_r$. Hence, any simplex of C' that has a facet on $C \cap h_r$ can serve as an initial simplex for h. We conclude that we can find initial simplices for all hyperplanes in $K(S)$ in time linear in the total size of the old conflict lists. (We may find more than one initial simplex for a hyperplane, but this is no problem.)

Once we have an initial simplex for each hyperplane $h \in K(S)$, we traverse the adjacency graph of the simplices to find the other simplices in S that are intersected by h. Since the subgraph of the adjacency graph induced by these simplices is connected, the total time spent in traversals, over all hyperplanes of $K(S)$, is linear in the total size of the new conflict lists.

The last thing we have to do in step 1 is to determine for each new simplex the level of one of its interior points with respect to $\mathcal{A}(H)$. To this end, we consider a simplex of C that was destroyed by h_r. Let p be the point interior to the destroyed simplex that defined its level. Let Δ_p be the new simplex that contains p. Trivially, we now know the level of a point interior to Δ_p. To compute the level of an interior point for each of the other new simplices, we again traverse the adjacency graph. The traversal starts at Δ_p. When we step from one simplex to the next, we can update the level of an interior point in time proportional to the size of the conflict lists of the two simplices. Hence, the total time we need is linear in the total size of the conflict lists of the new simplices.

Testing active cells. Recall that ℓ_Δ denotes the level of an interior point of the simplex Δ. The level of any other point in Δ must lie in the range

$$I_\Delta = \ell_\Delta - w(\Delta), \ldots, \ell_\Delta + w(\Delta).$$

We call Δ *active* if I_Δ contains a level that we wish to compute. So if we are comput-

ing the $\leq k$-level, then Δ is active if $I_\Delta \cap \{0, 1, 2, \ldots, k\} \neq \emptyset$, and if we are computing the k-level, then Δ is active if $k \in I_\Delta$. Since we are maintaining a subcomplex of $\mathcal{A}(R)$, we cannot discard individual simplices: a cell either has to be kept as a whole, or it has to be discarded as a whole. Of course, we should not discard cells that contain active simplices. Therefore we define a cell to be active if at least one of the simplices of its canonical triangulation is active. Note that after inserting the last hyperplane all conflict lists are empty, and so the remaining cell complex is exactly what we wish to compute. (If we are computing the k-level, the remaining complex consists of the cells of level k, from which the k-level can be reconstructed easily.)

Step 2 of our algorithm tests whether the new cells created by the insertion of h_r are active. To this end, we store with each cell a counter indicating the number of active simplices in its canonical triangulation. Let C be a cell that was split, and let C' be one of the two new cells. To compute the counter for C' we should only spend time on its new simplices and on the simplices from C that were destroyed; we should not spend time on simplices in C' that were already present in C. But this is easy given the counter for C and the levels of the new and old simplices. We conclude that we can test whether C' is active in time proportional to the number of new simplices in C' plus the number of destroyed simplices from C.

4. The analysis. We have seen that the total work for inserting h_r is proportional to the total size of the conflict lists of the simplices destroyed by h_r and of the newly created simplices. Since the simplices being destroyed must have been created before, the total work is proportional to $\sum_\Delta w(\Delta)$, where the summation is over all simplices created by the algorithm. Observe that it could happen that we create a new cell, spend time to triangulate it and to compute the conflict lists and the levels of all its simplices, and then immediately discard it. In other words, we may spend time on (simplices of) cells that are never active. Our analysis must take this into consideration, of course.

We are going to apply Lemma 2.2 to estimate the sum $\sum_\Delta w(\Delta)$ over all created simplices. The complex $\mathcal{K}_r$ maintained by the algorithm does not seem to be directly suitable for the analysis. We thus define for every $R \subseteq H$ an auxiliary complex $\bar{\mathcal{K}}(R)$, which can be used in the role of $\mathcal{C}(R)$ in Lemma 2.2.

For a cell C of $\mathcal{A}(R)$, its diameter $\operatorname{diam}(C)$ is defined as the maximum number of hyperplanes of H intersecting a segment fully contained in C. We define an auxiliary set $\bar{\mathcal{K}}_0(R)$ of d-cells of $\mathcal{A}(R)$. The algorithm for computing the $(\leq k)$-level includes a d-cell $C \in \mathcal{A}(R)$ into $\bar{\mathcal{K}}_0(R)$ if and only if it intersects $(\leq k')$-level in $\mathcal{A}(H)$, where $k' = k'(C) = k + 2(d+1)\operatorname{diam}(C) + d + 1$ (note that we take cells of the arrangement of R, but we consider levels in the arrangement of H). Similarly, the algorithm for computing the k-level includes a d-cell $C \in \mathcal{A}(R)$ in $\bar{\mathcal{K}}_0(R)$ if it intersects the region between the levels $k - 2(d+1)\operatorname{diam}(C) + d + 1$ and $k + 2(d+1)\operatorname{diam}(C) + d + 1$ of $\mathcal{A}(H)$. The complex $\bar{\mathcal{K}}(R)$ consists of the cells that belong to $\bar{\mathcal{K}}_0(R)$ or that share a facet with a cell of $\bar{\mathcal{K}}_0(R)$, plus their lower-dimensional faces.

LEMMA 4.1. *All simplices Δ created by the actual algorithm in the rth step are contained in $\bar{\mathcal{K}}(\{h_1, \ldots, h_r\})^\nabla \setminus \bar{\mathcal{K}}(\{h_1, \ldots, h_{r-1}\})^\nabla$; thus, a fictitious algorithm that maintains $\bar{\mathcal{K}}(R)^\nabla$ will create all simplices created by the actual algorithm.*

Proof. We prove the lemma for the algorithm that computes the k-level; the proof for the computation of the $\leq k$-level is analogous.

All cells created by the actual algorithm arise by splitting an active cell C. Let C' and C'' denote the two cells into which C is split. One of these two cells, say C', has diameter at least $(\operatorname{diam}(C) - 1)/2$. We shall prove that C' is in $\bar{\mathcal{K}}_0(\{h_1, \ldots, h_r\})$.

Since C' and C'' share a facet, we get that both C' and C'' lie in $\bar{\mathcal{K}}(\{h_1, \ldots, h_r\})$, from which the lemma follows.

Let Δ be an active simplex of C, and let p be the point in the interior of Δ defining its level ℓ_Δ. Since $w(\Delta) \le d \cdot \operatorname{diam}(C)$ and Δ is active, we know that

$$\ell_\Delta - d \cdot \operatorname{diam}(C) \le k \le \ell_\Delta + d \cdot \operatorname{diam}(C).$$

Let q be an arbitrary point in C'. The level of q differs from the level of p by at most $\operatorname{diam}(C)$. Hence,

$$\text{level of } q - (d+1)\operatorname{diam}(C) \le k \le \text{level of } q + (d+1)\operatorname{diam}(C),$$

and since $\operatorname{diam}(C') \ge (\operatorname{diam}(C) - 1)/2$, this proves the lemma. □

It is easy to check that the complex $\bar{\mathcal{K}}(R)$ satisfies the monotonicity condition $(*)$ in Lemma 2.2, so we may apply Lemma 2.2(ii) for $c = 1$ to $\bar{\mathcal{K}}(R)$. By Lemma 4.1 we can now bound the expected running time of our algorithm as follows.

COROLLARY 4.2. *Let f be a nondecreasing function such that $\mathbf{E}\left[|\bar{\mathcal{K}}(R)^\nabla|\right] \le f(r)$ for any $r = 1, 2, \ldots, n$, where the expectation is over a random choice of an r-element set $R \subseteq H$. Then the expected running time of the algorithm is*

$$O\left(\sum_{r=1}^{n} \frac{n}{r^2} f(r)\right).$$

An ε-net argument. We are going to estimate the function $f(r) = \mathbf{E}\left[|\bar{\mathcal{K}}(R)^\nabla|\right]$. General results of Haussler and Welzl [HW87] imply that, for a suitable constant $c = c(d)$, a random r-element sample $R \subseteq H$ has the following property with probability at least $1 - 1/r^d$: any line segment s that does not intersect any hyperplane of R intersects at most $c\frac{n}{r}\log r$ hyperplanes of H. (This is usually expressed by saying that R is an ε-net with respect to segments, with $\varepsilon = c\log r/r$ [HW87].) Let ε-NET(R) be a predicate expressing this property, that is, ε-NET(R) is true if and only if R has this property. We can write, using conditional expectations

$$\begin{aligned}\mathbf{E}\left[|\bar{\mathcal{K}}(R)^\nabla|\right] = {} & \mathbf{E}\left[|\bar{\mathcal{K}}(R)^\nabla| \;\middle|\; \varepsilon\text{-NET}(R)\right] \cdot \Pr[\varepsilon\text{-NET}(R)] \\ & + \mathbf{E}\left[|\bar{\mathcal{K}}(R)^\nabla| \;\middle|\; NOT\ \varepsilon\text{-NET}(R)\right] \cdot \Pr[NOT\ \varepsilon\text{-NET}(R)].\end{aligned}$$

Since $\bar{\mathcal{K}}(R)^\nabla$ has never more than $O(r^d)$ simplices, this is at most

$$(1 - 1/r^d) \cdot \mathbf{E}\left[|\bar{\mathcal{K}}(R)^\nabla| \;\middle|\; \varepsilon\text{-NET}(R)\right] + O(1). \tag{1}$$

Let us consider the case when ε-NET(R) is true. Let M be the maximum diameter of a cell in $\mathcal{A}(R)$; we thus have $M = O((n/r)\log r)$. Consider a cell C of $\bar{\mathcal{K}}(R)$. By definition of $\bar{\mathcal{K}}(R)$, C is adjacent to a cell C' that intersects the $\le(k+2(d+1)M+d+1)$-level of $\mathcal{A}(H)$ (this concerns the algorithm for $(\le k)$-level computation). Hence, C is completely contained in the $\le(k + 2(d+2)M + 2d + 1)$-level of $\mathcal{A}(H)$ (the constants could be improved by a more careful argument). Now set $\delta = 2(d+2)M + 2d + 1$. so all cells of $\bar{\mathcal{K}}(R)$ are contained in the $\le(k+\delta)$-level of $\mathcal{A}(H)$. Similarly, in the case of computing the k-level, all cells of $\bar{\mathcal{K}}(R)$ are contained in the region between levels $k - \delta$ and $k + \delta$. We denote the region that contains the cells of $\bar{\mathcal{K}}(R)$ in either algorithm by $T = T(r)$. Thus the quantity $|\bar{\mathcal{K}}(R)^\nabla|$ that we want to bound is at most proportional to the number of vertices of $\mathcal{A}(R)$ lying in the region T.

If R is a random r-element subset of H, then for any fixed vertex v of $\mathcal{A}(H)$ the probability of it being a vertex of $\mathcal{A}(R)$ is at most $(r/n)^d$. If R is conditioned to satisfy ε-NET(R), this probability can increase at most by the factor of $(1-r^{-d})^{-1}$. Hence, the expected number of vertices of $\mathcal{A}(R)$ in T is at most $(1-r^{-d})^{-1}(r/n)^d N_T$, where N_T stands for the number of vertices of $\mathcal{A}(H)$ in T. From equation (1) we get $f(r) = O(1 + N_T r^d/n^d)$. By Corollary 4.2, we can now bound the expected running time of the algorithm by

$$O(n) + \frac{O(1)}{n^{d-1}} \sum_{r=1}^{n} r^{d-2} N_T \,. \tag{2}$$

(This holds provided that f comes out as a nondecreasing function, which is the case in all our applications.)

Specific results. We can now bound the expected running time of our algorithm in various situations by substituting appropriate bounds on N_T.

First we look at the computation of the $\leq k$-level in three-dimensional space. Here N_T is bounded by the number of vertices of the $\leq(k+\delta)$-level, so $N_T = O(n(k+\delta)^2)$.

For the computation of the k-level, N_T is the total complexity of the levels $k-\delta$ to $k+\delta$. In the three-dimensional case we are mainly interested in sets of planes that are the image of a set of points in the plane under the transformation that maps the order-k Voronoi diagram of the points to the k-level of the planes. In this case all planes are tangent to the unit paraboloid and $N_T = O(n(k+\delta)\delta)$. For the k-level of a set of lines in the plane, we can use the following result by Dey [Dey97]: for any arrangement of n lines in the plane and for any integer $s < n-k$, the complexity of all j levels for $k \leq j \leq k+s$ is $O\left(n(k+s)^{1/3}s^{2/3}\right)$. In our situation $s = \delta$, so we obtain $N_T = O\left(n(k+\delta)^{1/3}\delta^{2/3}\right)$.

We summarize the results of these calculations in the following theorem.

THEOREM 4.3.

(i) *The $\leq k$-level in an arrangement of n planes in $\mathbb{R}^3$ can be computed in $O(nk^2 + n\log^3 n)$ expected time.*

(ii) *The kth order Voronoi diagram for n points in the plane can be computed in $O((n-k)k\log n + n\log^3 n)$ expected time by computing the k-level in the corresponding arrangement of n planes in $\mathbb{R}^3$.*

(iii) *The k-level in an arrangement of n lines in the plane can be computed in $O(nk^{1/3}\log^{2/3} n + n\log^2 n)$ expected time.*

Our algorithm also works in other situations, such as the computation of the $\leq k$-level of a set of lines, discs, or monotone curves in the plane. In these situations, however, there is an algorithm that obtains slightly better bounds. The details of this are described in the next section.

5. Improvements and extensions. The algorithm of the previous section is suboptimal when k is very small (polylogarithmic in n); in this case Mulmuley's algorithm is better. We can illustrate this on the algorithm for computing the $\leq k$-level in the plane, when k is a constant. The analysis of our algorithm accounts for the maintenance of the portion of $\mathcal{A}(R)$ roughly within the $\leq\delta$-level, where $\delta = O((n/r)\log r)$. According to this analysis we maintain a region of complexity $O(r\log r)$. Mulmuley's algorithm maintains the $\leq k$-level in the arrangement of the lines already inserted, which has only $O(r)$ complexity.

One simple improvement is, of course, to run Mulmuley's algorithm in parallel with ours and see which one finishes first. This eliminates the potential advantage of our algorithm—namely, that no peeling step is necessary—so it cannot be directly

applied to discs, say. We indicate another route to an improvement; currently, however, it only works for the planar case (which is not so interesting for lines, since an optimal algorithm has been known there). We first explain the approach for the construction of the $\leq k$-level in an arrangement of lines in the plane. We then show that the improved algorithm also works for discs and curves.

Lazy clean-up and refined activity test. To improve the running time of the algorithm we should maintain fewer cells. In other words, the cells we maintain should be closer to the $\leq k$-level. To achieve this we will insist that each cell in the current complex $\mathcal{K}_r$ intersects the $\leq(k+2n/r)$-level (while the previous analysis used the fact that with high enough probability, each cell intersects the $\leq(k+O((n/r)\log r))$-level). There are two issues to be addressed. First, the new requirement is time dependent: a cell that was acceptable at some step r may have to be eliminated at some later step r', though it has not been split. Second, we have to refine the test of activity for a simplex: there can be simplices intersected by many more than n/r lines at step r, so we cannot use just one interior point to estimate the level of all points in such a simplex accurately enough.

To deal with the first issue, we use the so-called *lazy* strategy [dBDS94]. That is, we do not worry about cells that are not split but should be deactivated because r has increased. Thus, when we insert a new line we only perform an activity test for the newly created cells. Of course, cells that should be eliminated cannot be kept around for too long. We discard these cells at periodic clean-up steps. In particular, we do a clean-up after steps $1, 2, 4, 8$, and so on. Since we do only a logarithmic number of clean-ups, we can afford to do them in a brute-force manner: we traverse the entire current complex and eliminate all cells that do not contain a point of the $\leq(k+n/r)$-level, performing an activity test described below for each cell and discarding those which do not pass the test. In this way, we make sure that at any step r each cell intersects the $\leq(k+2n/r)$-level.

Now we describe the activity test for a cell C at step r. Again, we test each simplex $\Delta \in C^{\nabla}$ separately. We subdivide the simplex into smaller simplices, each intersected by at most n/r lines of $H \setminus R$, and we determine the level of an interior point for each of these small simplices. Now Δ is active if and only if at least one of these points is at level less than or equal to $k + n/r$. This test is again conservative: cells that contain a part of the $\leq k$-level are always active. Subdividing Δ into smaller simplices plus determining their levels can be performed in $O(w(\Delta)^2 r/n)$ expected time using a randomized algorithm of Chazelle and Friedman [CF90] (see also [Mat91]). At a regular insertion step, this refined activity test is performed only for the newly created cells. At the clean-up steps the test is performed for all the cells. This implies that a cell that is present at step r must have been tested (and found active) at some step r' with $r/2 \leq r' \leq r$. Hence, a cell that is present at time r (that is, after the insertion of the rth line has been completed) contains a point of level at most $k + 2n/r$. This finishes the description of the modifications to the algorithm.

More general ranges. The modified algorithm (and also the basic algorithm) can be applied to arrangements of ranges other than halfspaces. We only need to supply a suitable notion of a canonical triangulation (a subdivision of the arrangement into constant complexity cells) and implement the steps of the algorithm suitably. As an illustration, we mention two planar cases, namely discs and x-monotone Jordan curves with a bounded number of intersections for every pair of curves.

As a canonical triangulation, we use the *vertical decomposition* of the cells: we extend a vertical segment upward from every intersection point of two curves until it hits another curve and extend a vertical segment downward until it hits another

curve; if a vertical segment does not intersect any curve, it is extended to infinity. We call the constant complexity cells arising in this decomposition the *trapezoids*. The vertical decomposition can be updated is the same way as in randomized incremental algorithms for computing full planar arrangements of curves [Mul91a]. For the refined activity test we need one more ingredient: we must be able to decompose a trapezoid into smaller trapezoids, each intersected by at most n/r curves. An obvious modification of the Chazelle and Friedman algorithm [CF90] can compute such a subdivision in $O(w(\Delta)^2 r/n)$ expected time. Now we have all the tools available to implement the algorithm described above (or the basic one of the previous section).

The analysis. To analyze the modified algorithm we again define an auxiliary, possibly larger complex $\widetilde{\mathcal{K}}(R)$: a cell $C \in \mathcal{A}(R)$ belongs to $\widetilde{\mathcal{K}}(R)$ if it has a point of level $\leq(k+2n/r)$ or is adjacent to a cell with such a point, where $r = |R|$. The complex $\widetilde{\mathcal{K}}(R)$ has the monotonicity property $(*)$ needed in Lemma 2.2. There is also an analog of Lemma 4.1, namely, that all cells created in the rth step of the algorithm are in $\widetilde{\mathcal{K}}(\{h_1, \ldots, h_r\}) \setminus \widetilde{\mathcal{K}}(\{h_1, \ldots, h_{r-1}\})$, and all cells present in the actually maintained collection in the rth step are in $\widetilde{\mathcal{K}}(\{h_1, \ldots, h_r\})$.

The time needed for the insertion steps (not counting the clean-up phases) is bounded by $\sum_\Delta w(\Delta)^2 r(\Delta)/n$, where we sum over all simplices created by the algorithm, and where $r(\Delta)$ is the moment of creation of Δ. Using Lemma 2.2(ii) for $c = 2$ we get the bound of

$$O\left(\sum_{r=1}^{n} \frac{n}{r^2} f(r)\right), \tag{3}$$

where $f(r)$ is a nondecreasing upper bound for $\mathbf{E}\left[|\widetilde{\mathcal{K}}(R)^\nabla|\right]$.

The clean-up phases can be accounted for as follows. The simplices that are discarded in a clean-up phase after inserting the rth line passed an activity test at some time $r' \geq r/2$, so the time for the activity test that discards them can be charged to that previous test. Every test gets charged at most once this way. The simplices that pass the activity test at the clean-up phase after step r belong to $\widetilde{\mathcal{K}}(R)^\nabla$, $R = \{h_1, \ldots, h_r\}$, and hence the time for these tests at the considered clean-up phase is at most $\sum_{\Delta \in \widetilde{\mathcal{K}}(R)^\nabla} (w(\Delta)^2 r/n)$. Applying Lemma 2.2(i) with $c = 2$ and summing up over all clean-up phases, we get a contribution of $\sum_{i=1}^{\lfloor \log_2 n \rfloor} (n/2^i) f(2^i)$, which will be of the same order as equation (3).

It remains to provide a good bound on the expected size of $\widetilde{\mathcal{K}}(R)^\nabla$. This is at most proportional to the expected number of vertices of $\widetilde{\mathcal{K}}(R)$. We could use an ε-net argument again, arguing that, with high probability, all such vertices are at level at most $k + O((n/r)\log r)$ in the arrangement of H. This, however, gives nothing better than the analysis of the basic algorithm. In the planar case, a refined argument can be given (although the resulting bound is probably still not tight). We start with the case of lines.

First, we pick a number k' in the range $[k + 2n/r, 2(k + 2n/r)]$ such that the k'-level in $\mathcal{A}(H)$ has $O(n)$ complexity (this is possible since the total complexity of all levels in this range is $O(n(k + n/r))$). Then we divide the vertices of $\widetilde{\mathcal{K}}(R)$ into two types: those within the $\leq k'$-level of H and those outside. The expected number of vertices of the first type is $O((r/n)^2 nk')$. As for the vertices of the second type, they all belong to cells of $\mathcal{A}(R)$ intersecting L, the k'-level of $\mathcal{A}(H)$, or the adjacent cells. The collection of such cells can naturally be called the 1-*zone* of L. It follows from the probabilistic argument of Clarkson and Shor [CS89] that the complexity of

the 1-zone is of the same order as the complexity of the *zone* of L, that is, of the cells intersecting L, and it remains to bound the expected complexity of this zone. Let m denote the number of intersections of the lines of R with L (a contiguous segment is counted as one intersection). The number of intersections of the lines of H with L is $O(n)$ (each line meeting L does so at a vertex), and hence the expectation of m is $O(r)$. We may now use a standard trick to convert the zone of L to a single cell in an arrangement of $O(m+r)$ (possibly unbounded) segments, namely, we remove from each line of R a small portion around its intersections with L. By [GSS89], the complexity of a single cell in an arrangement of n segments is $O(n\alpha(n))$, and so the expected zone complexity in our case is $O(r\alpha(n))$. We may thus conclude that $f(r) = O(r^2k/n + r\alpha(n))$, and the expected running time of the algorithm is $O(nk + n\log n\,\alpha(n))$.

The same analysis can be applied for other ranges as long as we can bound the expected complexity of the zone of the k'-level. For the case of discs or x-monotone Jordan curves, the zone complexity can be bounded using the same argument, since one has a good bound on the complexity of a single cell in an arrangement of the corresponding (curvilinear) segments: a single cell defined by n segments of curves with at most s intersections per pair has complexity $O(\lambda_{s+2}(n))$ [GSS89]. We obtain the following result.

THEOREM 5.1. *The $\le k$-level in an arrangement of n discs in the plane can be computed in expected $O(nk + \lambda_4(n)\log n)$ time. The $\le k$-level in an arrangement of n x-monotone Jordan curves with each pair having most s intersections can be computed in expected $O(k^2\lambda_s(n/k) + \lambda_{s+2}(n)\log n)$ time.*

6. Concluding remarks. We presented randomized incremental algorithms for computing $\le k$-levels or k-levels in arrangements. The main difference from previous algorithms is that we use estimates of the level in the final arrangement to decide which part of the current arrangement to keep, whereas previous algorithms used the level in the current arrangement. Consequently, our algorithm maintains less information (unless k is very small) and therefore it is faster. For most values of k we have shown that our algorithm is optimal. We think that it is also optimal for small k, but we have not been able to prove this.

This is one of the questions we leave open: is it possible to tighten the analysis of the basic algorithm and show that it is optimal for any k (both for computing the $\le k$-level and for computing the kth order Voronoi diagram in the plane)? If one cannot do this for the basic algorithm, then maybe it is possible for the modified algorithm. Here one has a purely combinatorial question: what is the expected complexity of the zone of $L_{k'}$ in the arrangement of a random sample of size r, where $L_{k'}$ is the k'-level in the full arrangement? (Here k' can be chosen suitably so that the k'-level has no more than the average complexity.) It seems quite likely that this expectation should be $O(r)$ in the plane and $O(r^2k'/n)$ in 3-space.

REFERENCES

[AAS97] P. K. AGARWAL, B. ARONOV, AND M. SHARIR, *On levels in arrangements of lines, segments, planes, and triangles*, in Proc. 13th Annual ACM Sympos. Comput. Geom., ACM, New York, 1997, pp. 30–38.

[ABFK92] N. ALON, I. BÁRÁNY, Z. FÜREDI, AND D. KLEITMAN, *Point selections and weak ε-nets for convex hulls*, Combin. Probab. Comput., 1 (1992), pp. 189–200.

[AMS97] P. K. AGARWAL, J. MATOUŠEK, AND O. SCHWARZKOPF, *Computing many faces in arrangements of lines and segments*, SIAM J. Comput., 27 (1998), pp. 493–507.

[AS92] F. Aurenhammer and O. Schwarzkopf, *A simple on-line randomized incremental algorithm for computing higher order Voronoi diagrams*, Internat. J. Comput. Geom. Appl., 2 (1992), pp. 363–381.

[ASS89] P. K. Agarwal, M. Sharir, and P. Shor, *Sharp upper and lower bounds on the length of general Davenport-Schinzel sequences*, J. Combin. Theory Ser. A, 52 (1989), pp. 228–274.

[BDT93] J.-D. Boissonnat, O. Devillers, and M. Teillaud, *A semidynamic construction of higher-order Voronoi diagrams and its randomized analysis*, Algorithmica, 9 (1993), pp. 329–356.

[CE87] B. Chazelle and H. Edelsbrunner, *An improved algorithm for constructing kth-order Voronoi diagrams*, IEEE Trans. Comput., C-36 (1987), pp. 1349–1354.

[CEG+93] B. Chazelle, H. Edelsbrunner, L. Guibas, M. Sharir, and J. Snoeyink, *Computing a face in an arrangement of line segments*, SIAM J. Comput., 22 (1993), pp. 1286–1302.

[CF90] B. Chazelle and J. Friedman, *A deterministic view of random sampling and its use in geometry*, Combinatorica, 10 (1990), pp. 229–249.

[Cha93] B. Chazelle, *Cutting hyperplanes for divide-and-conquer*, Discrete Comput. Geom., 9 (1993), pp. 145–158.

[Cla87] K. L. Clarkson, *New applications of random sampling in computational geometry*, Discrete Comput. Geom., 2 (1987), pp. 195–222.

[Cla88] K. L. Clarkson, *A randomized algorithm for closest-point queries*, SIAM J. Comput., 17 (1988), pp. 830–847.

[CS89] K. L. Clarkson and P. W. Shor, *Applications of random sampling in computational geometry*, II, Discrete Comput. Geom., 4 (1989), pp. 387–421.

[CSY87] R. Cole, M. Sharir, and C. K. Yap, *On k-hulls and related problems*, SIAM J. Comput., 16 (1987), pp. 61–77.

[dBDS94] M. de Berg, K. Dobrindt, and O. Schwarzkopf, *On lazy randomized incremental construction*, Discrete Comput. Geom., 14 (1995), pp. 161–186.

[Dey97] T. Dey, *Improved bounds for planar k-sets and related problems*, in Proc. 38th Annual IEEE Symposium on the Foundations of Computer Science, IEEE Computer Society Press, Miami, FL, 1997, pp. 156–161.

[DE93] T. Dey and H. Edelsbrunner, *Counting triangle crossings and halving planes*, Discrete Comput. Geom., 12 (1994), pp. 281–289.

[Ede87] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Heidelberg, 1987.

[ELSS73] P. Erdős, L. Lovász, A. Simmons, and E. Straus, *Dissection graphs of planar point sets*, in A Survey of Combinatorial Theory, J. N. Srivastava, ed., North-Holland, Amsterdam, 1973, pp. 139–154.

[ERvK96] H. Everett, J.-M. Robert, and M. van Kreveld, *An optimal algorithm for the ($\leq k$)-levels, with applications to separation and transversal problems*, Internat. J. Comput. Geom. Appl., 6 (1996), pp. 247–261.

[EW86] H. Edelsbrunner and E. Welzl, *Constructing belts in two-dimensional arrangements with applications*, SIAM J. Comput., 15 (1986), pp. 271–284.

[GSS89] L. J. Guibas, M. Sharir, and S. Sifrony, *On the general motion planning problem with two degrees of freedom*, Discrete Comput. Geom., 4 (1989), pp. 491–521.

[HW87] D. Haussler and E. Welzl, *Epsilon-nets and simplex range queries*, Discrete Comput. Geom., 2 (1987), pp. 127–151.

[Lov71] L. Lovász, *On the number of halving lines*, Ann. Univ. Sci. Budapest, Sect. Math., 14 (1971), pp. 107–108.

[Mat91] J. Matoušek, *Cutting hyperplane arrangements*, Discrete Comput. Geom., 6 (1991), pp. 385–406.

[Mul91a] K. Mulmuley, *A fast planar partition algorithm*, II, J. ACM, 38 (1991), pp. 74–103.

[Mul91b] K. Mulmuley, *On levels in arrangements and Voronoi diagrams*, Discrete Comput. Geom., 6 (1991), pp. 307–338.

[Mul93] K. Mulmuley, *Computational Geometry: An Introduction Through Randomized Algorithms*, Prentice Hall, New York, 1993.

[PSS92] J. Pach, W. Steiger, and E. Szemerédi, *An upper bound on the number of planar k-sets*, Discrete Comput. Geom., 7 (1992), pp. 109–123.

[Sha91] M. Sharir, *On k-sets in arrangements of curves and surfaces*, Discrete Comput. Geom. 6 (1991), pp. 593–613.

[ZV92] R. Živaljević and S. Vrećica, *The colored Tverberg's problem and complexes of injective functions*, J. Combin. Theory Ser. A, 61 (1992), pp. 309–318.

SIAM J. COMPUT.
Vol. 27, No. 3, pp. 668–681, June 1998
© 1998 Society for Industrial and Applied Mathematics
005

A CONSTANT TIME OPTIMAL PARALLEL ALGORITHM FOR TWO-DIMENSIONAL PATTERN MATCHING*

MAXIME CROCHEMORE†, LESZEK GĄSIENIEC‡, RAMESH HARIHARAN§, S. MUTHUKRISHNAN¶, AND WOJCIECH RYTTER‖

Abstract. We give an alphabet-independent deterministic parallel algorithm for finding all occurrences of a pattern array of size $m_h \times m_w$ in a text array of size $n_h \times n_w$ in the concurrent-read-concurrent-write–parallel-random-access-machine (CRCW–PRAM) model. Our algorithm runs in $O(1)$ time performing optimal, that is, $O(n_h \times n_w)$ work, following preprocessing of the pattern. This improves the previous best bound of $O(\log\log m)$ time with optimal work [A. Amir, G. Benson, and M. Farach, *Proceedings 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, ACM, New York, 1993, pp. 79–85], following preprocessing of the pattern, where $m = \max\{m_h, m_w\}$.

The preprocessing required by our algorithm (and that due to Amir, Benson, and Farach) can be accomplished in $O(\log\log m)$ time and $O(m_h \times m_w)$ work [M. Crochemore et al., manuscript, 1993], [R. Cole et al., manuscript, 1993].

Key words. pattern matching, PRAM, two-dimensional, witnesses, duelling, periodicity

AMS subject classifications. 68Q22, 68Q25

PII. S0097539795280068

1. Introduction.

The problem of *two-dimensional matching* (henceforth called 2D-matching) is to find all occurrences of a pattern array p of size $m_h \times m_w$ in a text array t of size $n_h \times n_w$. Without loss of generality, we assume that $m_w \geq m_h$. The text and the pattern are drawn from an alphabet set Σ. Our interest lies in designing an alphabet-independent parallel algorithm for this problem, that is, one with complexity not dependent on the alphabet size $|\Sigma|$. The one-dimensional matching problem, or the *string matching problem* as it is known, is that of finding all occurrences of a pattern string of length m in a text string of length n.

The early algorithms for sequential 2D-matching were based on multiple-pattern matching [Bi77, Ba78] or suffix trees [AL88] and were therefore alphabet-dependent. More precisely, these algorithms took time $O((n_h \times n_w + m_h \times m_w)\log|\Sigma|)$. The first alphabet-independent sequential algorithm for 2D-matching was obtained by Amir, Benson, and Farach [ABF92]. Their algorithm takes linear, i.e., $O(n_h \times n_w)$, time

*Received by the editors January 17, 1995; accepted for publication (in revised form) March 15, 1996. This paper combines work done by two independent groups comprising Crochemore, Gąsieniec, and Rytter and Hariharan and Muthukrishnan. A preliminary description of the results in this paper appears in the *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Los Alamitos, CA, 1993, pp. 248–258.

http://www.siam.org/journals/sicomp/27-3/28006.html

†Institute Gaspard Monge, University of Marne-la-Vallée, 93160 Noisy-le-Grand, France (mac@univ-mlv.fr). This author's research was partially supported by GDR AMI.

‡Instytut Informatyki, Uniwersytet Warszawski, Banacha 2, 02-097 Warszawa, Poland (lechu@mimuw.edu.pl). This author's research was partially supported by KBN grant 2-11-90-91-01 and European Community Cooperative Action IC-1000 (project ALTEC).

§Max-Planck Institut für Informatik, Saarbrücken (ramesh@csa.iisc.ernet.in). This author's research was done while at the Courant Institute, New York University and was supported by NSF grants CCR-8902221, CCR-8906949, CCR-9202900, and CCR-8901484.

¶Bell Labs, 2A-342, 700 Mountain Ave., Murray Hill, NJ 07974 (muthu@research.bell-labs.com). This author's research was done while at the Courant Institute, New York University and was supported by NSF/DARPA grant CCR-89-06949 and NSF grant CCR-91-03953.

‖Instytut Informatyki, Uniwersytet Warszawski, Banacha 2, 02-097 Warszawa, Poland (rytter@mimuw.edu.pl). This author's research was supported by KBN grant 8T11C01208.

following preprocessing of the pattern. They also showed how this preprocessing could be accomplished in an alphabet-dependent manner in $O((m_h \times m_w) \log |\Sigma|)$ time. Recently, Galil and Park [GP92] showed how this preprocessing can be accomplished in $O(m_h \times m_w)$ time, independent of the alphabet size. Thus the sequential complexity of 2D-matching is $O(n_h \times n_h + n_w \times n_w)$ time, independent of the alphabet size. Crochemore and Rytter [CR95] give an easier exposition of this preprocessing.

Mathies [M88] gave a parallel deterministic algorithm for 2D-matching which took $O(\log^2 m_w)$ time using linear number of processors in the CRCW–PRAM model. Amir and Landau [AL88] gave an $O(\log m_w)$ time algorithm using $O(n_h \times n_w)$ processors in the CRCW–PRAM model. Both these algorithms are suboptimal. Karp and Rabin [KR87] obtained the first optimal $O(\log m_w)$ time algorithm for parallel 2D-matching on the exclusive-read-exclusive-write–parallel-random-access-machine (EREW–PRAM) model; however, their algorithm was a Monte Carlo type randomized algorithm. The first optimal deterministic algorithm was obtained by Kedem, Landau, and Palem [KLP89]. They used the technique of *naming* due to Karp, Miller, and Rosenberg [KMR72] to obtain an $O(\log m_w)$ time optimal algorithm for 2D-matching in the CRCW–PRAM model using quadratic space. (Also, see [CR91] for algorithms for 2D-matching based on naming.) Both these optimal algorithms for 2D-matching are alphabet-dependent. The first alphabet-independent optimal parallel 2D-matching algorithm was recently obtained by Amir, Benson, and Farach [ABF93]. They present two optimal algorithms, one that takes $O(\log \log m_w)$ time on the CRCW–PRAM, and another that takes $O(\log m_w)$ time on the concurrent-read-exclusive-write–parallel-random-access-machine (CREW–PRAM), both following preprocessing of the pattern. Their algorithms use linear space.

Our main result is a deterministic parallel CRCW–PRAM algorithm for 2D-matching which takes $O(1)$ time using $O(n_h \times n_w)$ processors, following preprocessing of the pattern. Our algorithm is alphabet-independent and uses linear space. This result is analogous to the recent result of Galil [Ga92] stating that string matching can be performed in $O(1)$ time and optimal work following preprocessing of the pattern. Indeed, our result is obtained by providing a simple constant time Turing-reduction from 2D-matching to string matching and utilizing the result in [Ga92]. Our result also yields a new linear time sequential algorithm for 2D-matching and a new parallel $O(\log m_w)$ time optimal algorithm for 2D-matching in the CREW–PRAM model.

Fast parallel algorithms for string matching rely on the concept of periodicity [Ga85] and witnesses [Vi85]. Similarly, the notions of two-dimensional periodicity developed by [AB92] have played a significant role in developing both sequential [ABF92, GP92] and parallel [ABF93] algorithms for 2D-matching. It is particularly interesting that in contrast to these algorithms, our algorithm was obtained without using any ideas regarding two-dimensional periodicity. Only a version of one-dimensional periodicity suitably generalized to two dimensions, which we term *h-periodicity* (or horizontal periodicity), is used. However, we use ideas from two recent parallel algorithms for string matching, namely, those in [Vi90] and [Ga92]. Our algorithm uses the notion of *deterministic samples*, introduced by Vishkin [Vi90] for string matching, suitably generalized to two dimensions. The constant time optimal string matching algorithm in [Ga92] is used as a black box. Our overall algorithm is surprisingly simple.

The crucial preprocessing in all alphabet-independent 2D-matching algorithms (including ours) is that of computing witnesses for the two-dimensional pattern. Recently it has been shown that witnesses can be computed in $O(\log \log m_w)$ time and

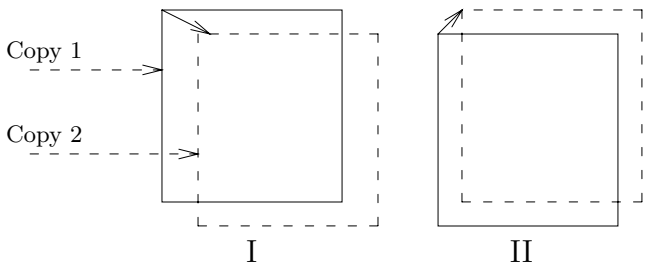


FIG. 1. *Quad* I *and* II *vectors.*

$O(m_h \times m_w)$ work [CG+93b]; this algorithm is both time and work optimal. Additionally, our algorithm (as the string matching algorithms of [Vi90, Ga92]) requires the computation of deterministic samples suitably generalized to two dimensions. This can be performed in $O(1)$ time optimally using the ideas in [CC+93], [CG+93a]. Thus the preprocessing required by our algorithm (and that in [ABF93]) takes $O(\log \log m_w)$ time and $O(m_h \times m_w)$ work.

Our algorithm is designed for the Common CRCW–PRAM model, i.e, simultaneous writes to the same location by several processors are guaranteed to be of the same value [Ja91]. Throughout this paper, the complexity bounds are stated in terms of time and work. Processor scheduling to attain these time and work bounds is always possible and hence omitted except where it is not obvious.

The paper is organized as follows. Section 2 gives some preliminary definitions. We describe important ideas underlying our algorithm in section 3. Section 4 describes the text processing algorithm.

2. Definition and preliminaries. Let p be the pattern array of size $m_h \times m_w$. Let t be the text array of size $n_h \times n_w$. Let $m' = \lfloor \frac{m_w}{3} \rfloor$ and $m'' = \lfloor \frac{m_h}{3} \rfloor$. Let $m = \max\{m_h, m_w\}$. Without loss of generality, assume that $m = m_w$. Let the column coordinates of p and t increase horizontally to the right and the row coordinates increase downwards, both starting from 1. The horizontal side of an array is called the *width* and its vertical side is called the *height*.

A pattern instance *associated* with text position l is a copy of p with top left corner at l; let p_l denote this pattern instance. We say that p *occurs* at l if p_l completely matches the text, i.e., if $l = (i, j)$, then $t(i+r-1, j+c-1) = p_l(r, c)$ for all $1 \leq r \leq m_h$ and $1 \leq c \leq m_w$. In this case, l is said to be an *occurrence* of p. The problem of *2D-matching* is to determine all occurrences of a given pattern p in a given text t.

Consider two copies of p. Place the second copy so that its left margin is either on or to the right of the left margin of the first copy. In addition, either the top left corner or the bottom left corner of the second copy must be within the first copy. The vector v joining the top left corner of the first copy to the top left corner of the second copy is called a Quad I vector in the former case and a Quad II vector in the latter case (see Figure 1). A *witness* for the two copies (or a witness for vector v) is a position in the first copy, if any, where the two copies mismatch. If a witness exists, then at least one of the two copies cannot match the text completely (if the two copies are visualized to lie on the text). One of these two copies can thus be eliminated by simply comparing the character at the witness in the first copy with the overlapping text character. This process of eliminating one of two overlapping copies of p possessing a witness is called a *duel* [Vi85] between the two copies.

If the two overlapping copies considered above are consistent, i.e., lack a witness, the vector joining the top left corners of these two consistent pattern copies is called a *period vector* of p. A period vector that is horizontal is called a *horizontal period vector* or *hpv* of p.

The *length* of a vector is either the difference between the row coordinates of its endpoints or the difference between the column coordinates of its endpoints, whichever is bigger. A vector is said to be *valid* if its row length is less than $\lfloor \frac{m_w}{3} \rfloor$ and its column length is less than $\lfloor \frac{m_h}{3} \rfloor$. If p has a valid *hpv*, we call p *h-periodic*. Otherwise, we call p *h-aperiodic*.

The *period* of a string s is defined to be $|s| - k$, where k is the length of the longest prefix of s which is also a suffix of s. s is said to be *periodic* if its smallest period is at most $\lfloor \frac{m}{3} \rfloor$ and *aperiodic* otherwise. Witnesses and duels are defined for strings in a manner analogous to the two-dimensional case.

The *distance* between two locations in a string is one more than the number of locations between them. This definition is extended to locations in the same row or column of a two-dimensional pattern in the natural way.

Recall that in the common CRCW–PRAM, given a boolean vector of i bits, their boolean AND and OR can be computed in $O(1)$ time using i processors. This fact, as well as the following lemma, are used repeatedly in our algorithm.

LEMMA 2.1 (see [FRW88]). *Given a binary vector V of size i, the leftmost or rightmost* 1 *in V can be found in $O(1)$ time using i processors on the common CRCW–PRAM.*

3. Horizontal periodicity. The following fact will be utilized repeatedly. It follows immediately from the definition of the shortest *hpv* of p.

FACT 3.1. *Any two occurrences of p with top left corners in some row of t must be at least distance l apart, where l is the length of the shortest hpv of p.*

Next, we state a crucial property of the h-aperiodic patterns.

LEMMA 3.2. *Suppose p is h-aperiodic. Then there exists a row r of p that, considered as a string, has period at least $\frac{\log m'}{2}$.*

Proof. The length of the smallest *hpv* of p is clearly the least common multiple (LCM) of the periods of the various rows of p considered as strings. By a weak form of the prime number theorem [RS62], the number of primes less than any number j is at most $2\frac{j}{\log j}$. Note that the LCM of the numbers $1, \ldots, j$ is the product of the highest powers at most j of all such primes. Therefore, the LCM of the numbers $1, \ldots, j$ is at most $j^{\frac{2j}{\log j}} \leq 4^j$. If all rows of p have period less than $\frac{\log m'}{2}$, then the length of the smallest *hpv* of p must be less than $4^{\frac{\log m'}{2}} \leq m'$. Since p has no such *hpv*, this is a contradiction. □

Next, we present the following two-dimensional variant of Vishkin's deterministic sampling lemma [Vi90] for strings.

LEMMA 3.3 (the two-dimensional deterministic sampling lemma). *Suppose p is h-aperiodic. For any $e \leq m'$, there exists a number f, $0 \leq f \leq e-1$, and a set of at most* $\log e$ *positions in p, called the (e, f)* h-sample*, with the following property. Let s be a text location and suppose all characters at locations in the h-sample of p_s match their aligned text characters. Then p cannot occur at any text location in the same row as s which is at most $e - f - 1$ locations to the left of s or at most f locations to the right of s.*

The proof of existence and the construction of an (e, f) *h-sample* for some suitable value of e are described in section 5.

4. Text processing. In what follows, we assume that witnesses in p, if any, for all valid vectors have been precomputed. We also assume that an (e, f) h-sample has been precomputed for $e = \lfloor (m')^{\frac{1}{4}} \rfloor$. In addition, we assume that the periods of all rows of p have been precomputed and the row r defined in Lemma 3.2 has been determined and preprocessed for matching it against any given text string using the algorithm in [Ga92]. In section 5, we consider the complexity of this precomputation.

4.1. *h*-aperiodic pattern. Assume that p is h-aperiodic. The case when p is h-periodic is a standard modification of this case and is described in section 4.2.

Definitions. Recall that p_l is the copy of p with top left corner at text position l. If p_l potentially matches the text, then l is a *source*. If p_l matches the text, then l is a *complete match*. A source l is *eliminated* if p_l is determined not to match the text. A source is said to *survive* at a given instant if it has not yet been eliminated. Sources k and l are *compatible* if p_k and p_l match wherever they overlap.

Initially, all text locations are sources. The algorithm eliminates sources until only complete matches survive. Consider the subarray t' of t of size $(n_h - m_h + 1) \times (n_w - m_w + 1)$ with the same top left corner as t. Clearly, pattern does not occur at any location outside t'. The subarray t' is divided into disjoint blocks of size $m'' \times m'$ (at the bottom and right boundaries of t', these blocks could be smaller than $m'' \times m'$). These blocks are called *text blocks*. There are $O(\frac{n_h \times n_w}{m_h \times m_w})$ text blocks. Each row in a text block is called a *text block row*.

Each text block T is processed identically in parallel in two phases. Phase I is survived only by a set of mutually compatible sources in T. In Phase II, each of these sources is either confirmed to be a complete match or is eliminated. Each phase takes $O(1)$ time and $O(m_h \times m_w)$ work per text block. The total work over all text blocks is $O(n_h \times n_w)$.

4.1.1. Phase I. Text block T is processed in two steps in this phase.

1. All but at most one of the sources in each text block row in T are eliminated.
2. Of the surviving sources, all but a set of mutually compatible sources are eliminated.

Each phase takes $O(1)$ time and $O(m_h \times m_w)$ work per text block.

We describe each step in detail.

Step 1. In each text block row in T, there cannot be more than one complete match since the pattern has no *hpv* of length less than m' (by Fact 3.1). In this step, all but at most one of the sources in each text block row in T are eliminated. This is achieved in the following two steps. All text block rows in T are considered in parallel in these steps. Consider one text block row $R[1, \ldots, m']$ in T.

Step 1.1. Following this step, at most $\frac{2m'}{\log m'}$ sources survive in R.

Let r be the row of p given by Lemma 3.2 and suppose that r is the ith row in p. Consider the text block row $R'[1, \ldots, m']$ that is $i - 1$ rows below R. Galil's algorithm [Ga92] is used to find all occurrences of r (considered as a string) beginning at positions in R'. This takes $O(1)$ time and $O(m' + m_w) = O(m')$ work. Since r has period at least $\frac{\log m'}{2}$ by Lemma 3.2, r occurs in R' beginning at, at most, $\frac{2m'}{\log m'}$ locations. Clearly, p occurs in R with its top left corner at $R[j]$ only if r occurs beginning at $R'[j]$, where $1 \leq j \leq m'$. Therefore, at most $\frac{2m'}{\log m'}$ sources in R survive this step.

Step 1.2. All but at most one of the sources in R are eliminated in this step.

This is done in several steps as follows. The (e, f) h-sample of p is used in the process, where $e = \lfloor (m')^{\frac{1}{4}} \rfloor$.

Step 1.2a. The text block row R is divided into disjoint subblocks of length $e = \lfloor (m')^{\frac{1}{4}} \rfloor$ (the rightmost subblock could be smaller). Following this step, at most two sources survive in each subblock.

Each surviving source in R is processed in parallel. Consider one source i and the pattern instance p_i associated with it. The set of locations in p_i which correspond to the (e, f) h-sample of p are considered. The characters at these locations are compared with their aligned text characters in parallel. This takes $O(1)$ time and $O(\frac{2m'}{\log m'} \times \log e) = O(m')$ work for R. We say that source i's sample *matches* if all the above comparisons are successful. Lemma 4.1 shows that the only sources in R which survive this step are the rightmost and leftmost sources in R whose sample matches in the above step. These sources are easily found in $O(1)$ time and $O(m')$ work using Lemma 2.1.

To perform the above step, $\lfloor \log e \rfloor$ processors have to be allocated to each surviving source in R from a pool of m' processors. This is done as follows. The text block row R is divided into disjoint subblocks of length $\lfloor \frac{\log m'}{2} \rfloor$. In each of these subblocks at most one source survives from Step 1.1. One processor is assigned to each of the m' locations in the text block row R, and the surviving source, if any, in each subblock, is determined. The surviving source in the kth subblock from the left in R is stored at location k in an auxiliary array A of size $\frac{m'}{\lfloor \log m'/2 \rfloor} = O(\frac{2m'}{\log m'})$. To each location in A, $\lfloor \log e \rfloor$ processors are assigned. The $\lfloor \log e \rfloor$ processors assigned to the location k in A are assigned to the surviving source in the kth subblock of R. That completes the assignment of $O(\frac{2m'}{\log m'} \times \log e) = O(m')$ processors in constant time.

LEMMA 4.1. *Source i in R survives Step* 1.2a *only if it is the rightmost or leftmost source in R whose sample matches.*

Proof. Clearly, if the sample of i does not match then p_i cannot be a complete match. Let j and k be the leftmost and rightmost sources in R whose samples match. Suppose the sample of i matches and i is between j and k. By Lemma 3.3, p cannot occur at any text location in R which is at most $e - f - 1$ locations to the left of k or at most f locations to the right of j. Since $(e - f - 1) + f > e - 2$, and i is in the middle $e - 2$ characters of R, p_i cannot be a complete match. □

Step 1.2b. The text block row R is divided into disjoint segments of length $\lfloor (m')^{\frac{1}{2}} \rfloor$ (the rightmost segment could be smaller). Following this step, at most one source survives in each segment.

All segments are processed in parallel. Consider a particular segment S. It comprises of $O(e)$ disjoint subblocks of length e from Step 1.2a. Each of these subblocks contains at most two surviving sources. In all, there are at most $O(2e)$ surviving sources in S. All pairs of surviving sources in S are duelled in parallel in $O(1)$ time and $O((2e)^2) = O(\sqrt{m'})$ work. Over all segments, the work done is $O(m')$. A source in S survives this step if and only if it survives each of these duels. Clearly, only one source in S survives all duels since p is h-aperiodic (by Fact 3.1).

Step 1.2c. At most one source survives in each block row R following this step.

Following Step 1.2b, there are $O(\sqrt{m'})$ disjoint segments of length $\lfloor \sqrt{m'} \rfloor$ in R, each of which contains at most one surviving source. In all there are at most $O(\sqrt{m'})$ surviving sources in R. Each pair of surviving sources in R is duelled; this takes $O(1)$ time and $O(m')$ work. A source survives this phase if and only if it survives each of these duels. Clearly, only one source in R survives all duels since p is h-aperiodic (by Fact 3.1).

Step 2. All text blocks are processed in parallel. Consider a particular text block T. Only sources in T that are mutually compatible survive this step.

Following Step 1, at most one source per text block row in T survives. In all there are at most m'' sources in T. Each pair of surviving sources is duelled in parallel; this takes $O(1)$ time and $O(m'' \times m'')$ work. Recall that $m_w \geq m_h$ and $m'' = \lfloor \frac{m_h}{3} \rfloor$. Therefore, the work done in this step is $O(m_h \times m_w)$. A source in T survives this phase if and only if it survives each of these duels. Clearly, all sources in T that survive this phase are mutually compatible.

That completes the description of Phase I. There are $O(\frac{n_h \times n_w}{m_h \times m_w})$ blocks in all and each step for each text block takes $O(1)$ time and $O(m_h \times m_w)$ work. Therefore Phase I takes $O(1)$ time and $O(n_h \times n_w)$ work over the entire text.

4.1.2. Phase II. In Phase II, each surviving source in each text block is either eliminated or confirmed to be a complete match. Each text block is considered in parallel. We describe Phase II for a particular text block T.

Consider the set of text locations overlapped by at least one of the p_i's, where i is a surviving source in T. There are at most $O(m_h \times m_w)$ such text locations, all of which lie in a text block T' of size $(m'' + m_h - 1) \times (m' + m_w - 1)$ whose top left corner coincides with the top left corner of T. From Phase I, all surviving sources in T are mutually compatible. Therefore, each text location in T' is overlapped by the same character, if any, in all p_i's. Hence, it suffices to compare each text location x in T' with the overlapping character in any of the p_i's which overlap x.

We say that a text location x in T' *belongs to* source y in T if x is overlapped by p_y. Phase II proceeds in three steps.

1. Each text location x in T' is marked with a source y in T, if any, to which x belongs.
2. Each text location x in T' which is marked with a source y is compared with the character in p_y which overlaps x.
3. For each surviving source y in T, if a mismatch occurs in Step 2 at a text location x overlapped by p_y, then y is eliminated.

We describe the three steps in Phase II in detail.

Step 1. This step marks each text location x in T' with a source y in T to which it belongs. Initially, none of the text locations in T' is marked. There are two substeps.

Step 1a. Each of the m' leftmost columns of B' are processed in parallel. Consider a particular column c. The uppermost and lowermost surviving sources, u and l, in c are determined. This is done in $O(1)$ time and $O(m_h)$ work using Lemma 2.1. Each text location x in c which is above l but below u (including u) is marked with source u. Each text location x in c which is at most distance $m_h - 1$ below l (including l) is marked with source l.

Step 1b. All rows of T' are processed in parallel. Consider row R. Let R' denote the leftmost m' locations of R. The leftmost and rightmost text locations in R' which were marked in Step 1a are determined in $O(1)$ time and $O(m_w)$ work. Let them be denoted by a and b, respectively. The text locations in R are marked as follows. Each unmarked text location in R which is to the left of b but to the right of a is marked with the same source as a. Each unmarked text location in R which is at most distance $m_w - 1$ to the right of b is marked with the same source as b.

LEMMA 4.2. *A text location x in T' is marked with exactly one of the sources, if any, to which it belongs.*

Proof. It can be easily seen that if x is marked with source y in Steps 1a or 1b, then x belongs to y. We show that if x belongs to some source in T, then it is marked with one such source. We consider two cases.

First, suppose x belongs to some source in the same column as x. Let u and l denote the uppermost and lowermost surviving sources in this column. If x is above l, then it must be coincident with or below u; it is then marked with source u in Step 1a. Otherwise, if x is coincident with or below l then it is at most distance $m_h - 1$ below l; it is then marked with source l.

Second, suppose x does not belong to any source in the same column c as x but belongs to some source in a column to the left of c. In Step 1a, x will not be marked as all sources in the same column, as x must be the distance of at least m_h above x. Let a and b be the leftmost and rightmost locations in the same row as x which were marked in Step 1a. Since x belongs to some source in T to the left of c, a and b are defined by the first argument above, although they are not necessarily distinct.

Let $s(a)$ and $s(b)$ denote the sources with which a and b are marked, respectively. Note that $s(a)$ is in the same column as a and likewise for $s(b)$. None of the locations to the right of b in the same row as x is marked in Step 1a. By the first case, none of these locations belongs to any of the sources in their respective columns. It follows that x does not belong to any of the sources in these columns. Since x does belong to some source in T, either x is to the left of b and coincident with or to the right of a, or x is within distance $m_w - 1$ to the right of b. In the former case, x belongs to $s(a)$, and in the latter case, x belongs to $s(b)$. In Step 1b, x is marked with $s(a)$ in the former case and with $s(b)$ in the latter case. □

Step 2. Consider each text location x in T' marked with a source y. Then the character at x is compared with the overlapping character in p_y.

A text location that mismatches in Step 2 is called a *bad* position.

Step 3. Note that a mismatch at text location x in Step 2 eliminates all sources in a $m_h \times m_w$ block whose bottom right corner is x. A second kind of marking is done in this step. Initially none of the locations in T' is marked. Finally, for each bad text location x, all text locations in T' which are in a $m_h \times m_w$ block with bottom right corner at x are marked. This is done in two steps.

Step 3a. All rows of T' are considered in parallel. Consider row R of T'. We divide R into two parts; the first part R_1 consists of the leftmost m_w text locations and the second part R_2 consists of the remaining locations. The rightmost and leftmost bad text locations a and b, respectively, in R_1, and the rightmost and leftmost bad text locations c and d, respectively, in R_2 are determined. This is done in constant time and $O(m_w)$ work using Lemma 2.1.

Every text location that is in R and at most distance $m_w - 1$ to the left of either a or c is marked, as are those text locations between a and b and between c and d.

Step 3b. All columns of T' are considered in parallel. Consider a particular column C of T'. We divide C into two parts; the first part C_1 consists of the top m_h text locations and the second part C_2 consists of the remaining locations. The lowermost and uppermost marked text locations a and b, respectively, in C_1, as well as the lowermost and uppermost marked text locations c and d, respectively, in C_2 are determined in constant time and $O(m_h)$ work using Lemma 2.1. Every text location which is in C and which is at most distance $m_h - 1$ above either a or c is marked. All text locations between a and b and between c and d are also marked.

LEMMA 4.3. *A surviving source y in T is marked if and only if there exists a bad location x which belongs to y.*

Proof. The argument proceeds as in the proof of Lemma 4.2. In Step 3a, a mark is placed on a text location in T' if and only if it is at most distance $m_w - 1$ to the left of some bad text location in the same row. In Step 3b, a mark is placed on each text location in T which is at most distance $m_h - 1$ above some text location in the

same column that was marked in Step 3a. From these two claims, the lemma follows as in the proof of Lemma 4.2. □

Finally, all surviving sources in T which are not marked are declared to be complete matches in Step 3. It follows from Lemma 4.3 that these are the only complete matches. Each step in Phase II takes $O(1)$ time and $O(n_h \times n_w)$ work. That completes the description of the algorithm for h-aperiodic patterns.

4.2. *h*-periodic patterns. Our approach for this case is essentially an adaptation of the approaches in [BG90, Vi90] to two-dimensional patterns and is described here for the sake of completeness.

Say the smallest *hpv* of p has length l, $l < \frac{m_w}{3}$. Then $p = u^k v$ where u comprises the leftmost l columns in p, v comprises the ($m_w \bmod l$) rightmost columns of p, and $k = \lfloor \frac{m_w}{l} \rfloor \geq 3$. In addition, the ($m_w \bmod l$) leftmost columns of u and p are identical to v. Define p' to be uuv.

LEMMA 4.4. *p' is h-aperiodic and l is the length of its shortest hpv.*

Proof. Clearly, p' has an *hpv* of length l. Suppose for a contradiction that p' also has an *hpv* of length $l' < l$. Then, by a straightforward generalization of the GCD lemma [LS62], p' has an *hpv* of length $\gcd(l', l) = l'' < l$. Clearly, l'' divides l and therefore $u = u'^{k'}$, where u' comprises the first l'' columns of u. It follows that p has an *hpv* of length $l'' < l$, which is a contradiction. □

In order to find all occurrences of p in t, all occurrences of p' in t are found as described in section 4.1. Given all occurrences of p', all occurrences of p are determined in $O(1)$ time and $O(n_h \times n_w)$ work as described below.

The rows of the text are divided into disjoint blocks of length $\lfloor \frac{m_w}{4} \rfloor$ (the rightmost block in each row could be smaller). All such blocks are processed in parallel. Occurrences of p with top left corner in each such block are found in $O(1)$ time and $O(m_w)$ work per block. We describe the procedure for one such block B.

For a location x in B, $x + l$ is defined to be the location which is distance $l - 1$ to the right of x in its row. Given two locations a and b in the same row of the text, b to the right of a, let $b - a$ denote the distance between a and b. The following steps are performed.

Step 1. The rightmost location x in B such that p' matches the top left corner at x is determined in $O(1)$ time and $O(m_w)$ work using Lemma 2.1.

Step 2. The leftmost location y in B such that $x - y$ is a multiple of l and p' matches the top left corner at $y, y + l, y + 2l, \ldots, x$ is determined in $O(1)$ time and $O(m_w)$ work using Lemma 2.1. Similarly, the rightmost location z at most distance $m_w - 1$ to the right of x such that $z - x$ is a multiple of l and p' matches the top left corner at $x, x + l, x + 2l, \ldots, z$ is determined in $O(1)$ time and $O(m_w)$ work using Lemma 2.1. Note that z need not be in B.

Step 3. All occurrences of p with top left corner in B are found using Lemma 4.5 in $O(1)$ time and $O(m_w)$ work.

LEMMA 4.5. *Suppose p' matches the text with top left corner at location a in B. p_a matches the text if and only if $z - a \geq (k - 2)l$ and a is either coincident with or to the right of y.*

Proof. First, suppose a is either coincident with or to the right of y and $z - a \geq (k - 2)l$. Since p' occurs with top left corner at $y, y + l, \ldots, x$ and since l is the length of the shortest *hpv* of p', $a - y$ is a multiple of l. Further, the right boundary of p_a is aligned with or to the left of the right boundary of the copy of p' with top left corner at z. It follows that there is an occurrence of u with top left corner at $a + l, a + 2l, \ldots, a + (k - 1)l$ and an occurrence of v at $a + kl$. Therefore, p_a matches the text.

Second, suppose either a is to the left of y or a is coincident with or to the right of y and $z - a < (k-2)l$. We consider the two cases separately.

Suppose a is coincident with or to the right of y and $z - a < (k-2)l$. As shown in the previous paragraph, $a - y$ is a multiple of l. For p_a to match the text, p' should occur at $a+il$, for all i, $1 \le i \le k-2$. Since $z-a < (k-2)l$, there exists $i \le k-2$ such that $a+il = z+l$. So p_a matches the text only if p' occurs with top left corner at $z+l$. By the definition of x, $z - x < (k-2)l$ and therefore $z + l - x \le (k-2)l \le m_w - 1$. By the definition of z, p' cannot occur with its top left corner at $z + l$ and therefore p_a does not match the text.

Next, suppose a is to the left of y in B. Let h be the smallest number such that $a+hl \ge y$. If $a+hl > y$ and p' occurs with top left corner at $a+(h-1)l$ then, since p' also occurs with top left corner at y, p' has an *hpv* of length smaller than l, which is a contradiction to Lemma 4.4. Therefore, either $a + hl = y$ or p' does not occur with top left corner at $a+(h-1)l$. This, along with the definition of y, implies that p' does not occur with top left corner at $a+(h-1)l$ in either case and $a+(h-1)l$ is to the left of y. Recall that y, a are in B and $k \ge 3$. Therefore, $(k-2)l \ge \frac{m_w}{4} \ge y - a \ge (h-1)l$ and $h - 1 \le k - 2$. For p_a to match the text, p' should occur at $a + il$, for all i, $1 \le i \le k-2$. Since p' does not occur with its top left corner at $a + (h-1)l$, p_a does not match the text. □

That completes the description of the entire algorithm and gives the following theorem.

THEOREM 4.6. *There is a deterministic CRCW–PRAM algorithm for 2D-matching which runs in $O(1)$ time performing $O(n_h \times n_w)$ work, following preprocessing of the pattern.*

Consider the complexity of 2D-matching on the CREW–PRAM model, that is, the PRAM model in which simultaneous writes are forbidden [Ja91]. Our 2D-matching algorithm uses the string matching algorithm due to Galil which utilizes concurrent writes. This can be replaced by the text processing of Vishkin's algorithm [Vi85] which takes $O(\log m)$ time and optimal work on a CREW–PRAM. In addition, in our 2D-matching algorithm, concurrent writes are used only to find either the boolean AND of the values in a boolean vector or the leftmost and rightmost 1's of a binary vector. These computations can easily be performed optimally on a CREW–PRAM with logarithmic time by computing along a balanced binary tree [Ja91]. That leads to the following theorem.

THEOREM 4.7. *There is a deterministic CREW–PRAM algorithm for 2D-matching which runs in $O(\log m_w)$ time performing $O(n_h \times n_w)$ operations, following preprocessing of the pattern.*

5. Preprocessing. We assume that p is h-aperiodic. If p is h-periodic then, as shown in section 4.2, only an h-aperiodic portion of p needs to be preprocessed; this h-aperiodic portion can be preprocessed similarly. Whether or not p is h-periodic can be determined in $O(1)$ time and $O(m_h \times m_w)$ work using the string witness computation algorithm of [BG90].

The following preprocessing is required by the text processing algorithm.

1. Witnesses, if any, for all valid Quad I and Quad II vectors of p.

The witnesses, if any, for valid Quad I and Quad II vectors of p are computed in $O(\log \log m)$ time and $O(m_h \times m_w)$ work [CG+93b].

2. The periods of each row of p considered as a string as well as the row r in Lemma 3.2.

Periods of each row are computed optimally in $O(\log \log m)$ [BG90, ABG92] time. Following this the row r can be identified in $O(1)$ time using m_h processors.

3. The precomputation required for Galil's string matching algorithm [Ga92] applied to row r (as defined in Lemma 3.2) of p.

This is accomplished in $O(\log\log m)$ time and $O(m_w)$ work [CG+93a].

4. The $(\lfloor (m')^{1/4} \rfloor, f)$ h-sample of p for some f, $0 \leq f \leq \lfloor (m')^{1/4} \rfloor - 1$.

Let $\alpha = \lfloor (m')^{1/4} \rfloor$. We show how to compute an (α, f) h-sample of p for some f, $0 \leq f \leq \alpha - 1$, in $O(1)$ time and $O(m_h \times m_w)$ work. This is done by a straightforward generalization of the algorithm of [CG+93a] for computing the deterministic sample for a string. We describe it here for the sake of completeness.

Definitions. Define an *α-block* to be a rectangle of width α and height m_h. B_i is defined to be the α-block comprising the columns $i, \ldots, i+\alpha-1$ of p, for $1 \leq i \leq \alpha$.

First, it is determined if B_i and B_j are identical for some i and j, $1 \leq i < j \leq \alpha$. This is done in $O(1)$ time and $O(m_h \times m_w)$ work by comparing all such pairs of α-blocks explicitly in parallel. There are $O((\alpha)^2)$ such pairs, comparing each of which takes $O(1)$ time and $O(m_h \times \alpha)$ work. In all, this step takes $O(1)$ time and $O(m_h \times (\alpha)^3) = O(m_h \times m_w)$ work. There are two cases next.

Case A. B_i and B_j are identical, for some i, j, $1 \leq i < j \leq \alpha$.

Let $j - i = c$.

Since p is h-aperiodic and $c < \alpha \leq m'$, there exists a column in p which differs from the cth column to its right. This column cannot be column i, as B_i and B_j are identical. Therefore, such a column must exist either to the left or right of column i. We consider two subcases next, depending on where this column lies.

Case A.1. There exists a column in p to the right of column i which differs from the cth column to its right. Let k be the leftmost such column.

Let l be the topmost row in which columns k and $k+c$ differ. Recall that B_i and B_j are identical. Note that if column k is in B_i, then column $k+c$ is in B_j and therefore columns k and $k+c$ are identical, which is a contradiction. It follows that column k is to the right of B_i, i.e., $k \geq i + \alpha$.

In this case, we claim that the (α, f) h-sample is $\{(l,k), (l,k+c)\}$, with $f = \alpha - 1$. This is shown as follows. In p, the characters at locations (l,k) and $(l,k+c)$ are different. In addition, by the definition of k, the characters at locations (l,k') and $(l,k'+c)$ are identical for all columns k', $i < k' < k$. Since $k \geq i + \alpha$, any copy p' of p shifted at most $\alpha - 1$ positions horizontally to the right of p has some k'th column aligned with column k in p, where $i < k' < k$. Since the characters at locations (l,k') and $(l,k'+c)$ are identical, p' differs from p at at least one of the positions $\{(l,k), (l,k+c)\}$.

Case A.2. Every column in p to the right of column i is identical to the cth column to its right. In addition, there exists a column in p to the left of column i which differs from the cth column to its right. Let k be the rightmost such column.

Let l be the topmost row in which columns k and $k+c$ differ. In this case, we claim that the (α, f) h-sample is $\{(l,k), (l,k+c)\}$, with $f = 0$. This is shown as follows. In p, the characters at locations (l,k) and $(l,k+c)$ are different. In addition, by the definition of k, the characters at locations (l,k') and $(l,k'+c)$ are identical for all columns k', $k' > k$. Since $k + \alpha < m_w$, any copy p' of p shifted at most $\alpha - 1$ positions horizontally to the left of p has some k'th column aligned with column k in p, where $k' > k$. Since the characters at locations (l,k') and $(l,k'+c)$ are identical, p' differs from p at at least one of the positions $\{(l,k), (l,k+c)\}$.

It remains to show how k and l can be computed in Cases A.1 and A.2. Each column is compared with the cth column to its right in parallel and the smallest row in which they differ, if any, is determined in $O(1)$ time and $O(m_h \times m_w)$ work. Following that, k and l can be computed easily in $O(1)$ time and $O(m_h \times m_w)$ work.

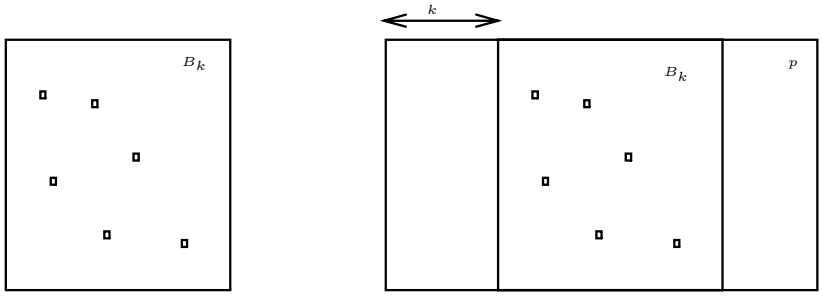


FIG. 2. *The sets $\mathcal{S}$ and $\mathcal{S}'$.*

This completes Case A.

Case B. B_i and B_j are different, for all i, j, $1 \leq i < j \leq \alpha$.

Consider the scenario in which $B_1, \ldots, B_\alpha$ are laid on top of each other with their boundaries aligned. We show how to compute a value k and a set $\mathcal{S}$ of at most $\log \alpha$ locations in B_k satisfying the following property: for each B_i, $i \neq k$, there exists $j \in \mathcal{S}$ such that B_i differs from B_k at location j, i.e., the character in B_i which overlaps location j in B_k differs from the character at location j in B_k. The required (α, f) h-sample of p, $f = k - 1$, is then determined as follows.

We claim that the h-sample of p is the set $\mathcal{S}'$ of locations in p which correspond to the locations in B_k which are in $\mathcal{S}$ (see Figure 2). This is shown as follows. If p' is any copy of p shifted horizontally to the right of p by i, $1 \leq i \leq k-1$, then the portion of p' which overlaps block B_k in p is identical to the block B_{k-i}. If p' is any copy of p shifted horizontally to the left of p by i, $1 \leq i \leq \alpha - k$, then the portion of p' which overlaps block B_k in p is identical to the block B_{k+i}. In each case, since B_k differs from all other α-blocks at some location in $\mathcal{S}$, p' differs from p at some location in $\mathcal{S}'$.

Computing k and $\mathcal{S}$. Let A be a two-dimensional table of size $\alpha \times (m_h \times \alpha)$ initialized to 0. The row i in A stands for B_i. Location (i, j) of A corresponds to the location (a, b) in B_i, where $a \in [1, \ldots, m_h]$, $b \in [1, \ldots, \alpha]$, $(a-1) \times m_h + b = j$; we refer to location (a, b) in B_i as location j in B_i.

The table A is filled in so as to satisfy the following criterion: $A(i, j) = 1$ if and only if for some $l \neq i$, $1 \leq l \leq \alpha$, j is the smallest number such that B_l differs from B_i at location j in B_i (note that the minimality of j signifies that B_l and B_i match at all columns to the left of the column containing location j in B_i and at all locations above location j in the column containing location j in B_i). This is done by exhaustively comparing all pairs of α-blocks in $O(1)$ time and $O((\alpha \times \alpha) \times (m_h \times \alpha)) = O(m_h \times \alpha^3) = O(m_h \times m_w)$ work. Here $\alpha \times \alpha$ is the number of pairs of α-blocks and $m_h \times \alpha$ is the size of each α-block.

Recall that all α-blocks are distinct; therefore, for all i, $1 \leq i \leq \alpha$, each B_l, $l \neq i$, differs from B_i at some location in B_i in the set $\mathcal{S}_i = \{j | A(i, j) = 1\}$. In Lemma 5.1, we show that there exists an f such that $|\mathcal{S}_f| \leq \log \alpha$. We also show that one such f can be found in $O(1)$ time and $O(m_h \times m_w)$ work. k is chosen to be the f so found and $\mathcal{S}$ is chosen to be $\mathcal{S}_f$.

LEMMA 5.1. *There exists an f, $1 \leq f \leq \alpha$ such that $|\mathcal{S}_f| \leq \log \alpha$.*

Proof. Recall that we are considering the scenario in which $B_1, \ldots, B_\alpha$ are laid on top of each other with their borders aligned. Also recall that all the B_i's are distinct.

The proof is constructive. The construction has at most $\log \alpha$ iterations. In each iteration, at least half of the α-blocks currently being considered are removed from further consideration until, finally, only one α-block remains. In iteration l, we also pick a number j_l. The following invariants hold at the end of iteration l.

1. All α-blocks B_i which remain under consideration after iteration l are identical at the locations $1, \ldots, j_l$.
2. For all α-blocks B_i which remain under consideration after iteration l, $A(i, j) = 1$ if and only if $j \in \{j_1, \ldots, j_l\}$, where $1 \leq j \leq j_l$.
3. If B_i is removed from consideration in iteration l, then it differs from all $B_{i'}$'s which are still under consideration at location j_l.

Let B_f be the only α-block which remains finally. We will show that $\mathcal{S}_f = \{j_1, \ldots, j_r\}$, where r is the number of iterations. The lemma follows immediately.

We describe the lth iteration. Let C be the set of α-blocks still under consideration. All α-blocks in C are identical at locations $1, \ldots, j_{l-1}$. j_l is chosen to be the smallest number greater than j_{l-1} with the property that $A(i, j_l) = 1$ for some $B_i \in C$ is chosen. Clearly, all α-blocks in C are identical at locations $1, \ldots, j_l - 1$, and two α-blocks in C differ at location j_l. Consider the multiset of characters which occur at location j_l in the α-blocks in C; there are at least two such characters. Let a be the character which appears the least number of times in this multiset. The set D of α-blocks which remain under consideration is comprised of all those α-blocks which have the character a in location j_l.

Clearly, $|D| \leq \lfloor \frac{|C|}{2} \rfloor$ and all α-blocks in D are identical at locations $1, \ldots, j_l$. Thus invariant 1 holds. In addition, if $B_i \in D$, then $A(i, j_l) = 1$. By the definition of j_l, $A(i, j) = 0$ for all $B_i \in D$ and $j_{l-1} < j < j_l$. Thus invariant 2 holds. For all $B_i \in C - D$ and $B_{i'} \in D$, B_i differs from $B_{i'}$ at location j_l. Thus invariant 3 holds as well.

It remains to show that $\mathcal{S}_f = \{j_1, \ldots, j_r\}$. Suppose for a contradiction that $\mathcal{S}_f \neq \{j_1, \ldots, j_r\}$. From invariant 2, it follows that there exists a $j > j_r$ such $A(f, j) = 1$. Then there exists B_i, $i \neq f$, such that B_i and B_f are identical at locations $1, \ldots, j_r$ but differ at location j. Since B_i was removed from consideration in some iteration l, B_l differs from B_f at location j_l by invariant 3, which is a contradiction. □

Finding a row in A with at most $\log \alpha$ 1's. It remains to determine one row in table A which contains at most $\log \alpha$ 1's. All rows are considered in parallel. For each row it is determined if it has at most $\log \alpha$ 1's, and in case it does, the locations containing 1's are compacted into an array of size $\log \alpha$. Lemma 5.2 shows that this can be done in $O(1)$ time and $O(m_h \times \alpha)$ work per row; this sums to $O(m_h \times \alpha^2) = O(m_h \times m_w)$ work over all rows.

LEMMA 5.2 (see [Ra90]). *Given a binary array X of size i and a parameter j, there is an algorithm which determines whether X has at most j 1's, and if so, compacts the indices containing 1's into an array of size j. This algorithm takes time $O(\log j / \log \log i)$ time and $O(i)$ work.*

This yields the following theorem.

THEOREM 5.3. *A two-dimensional pattern of size $m_h \times m_w$ can be matched against a two-dimensional text of size $n_h \times n_w$ in $O(1)$ time and $O(n_h \times n_w)$ work, after the pattern has been preprocessed in $O(\log \log m_w)$ time and $O(m_h \times m_w)$ work.*

Acknowledgments. We thank Richard Cole for his comments.

REFERENCES

[AB92] A. AMIR AND G. BENSON, *Two dimensional periodicity in rectangular arrays*, in Proceedings of the 3rd Annual ACM Symposium on Discrete Algorithms, ACM, New York, 1992, pp. 440–452.

[ABF92] A. AMIR, G. BENSON, AND M. FARACH, *Alphabet independent two dimensional matching*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, ACM, New York, 1992, pp. 59–68.

[ABF93] A. Amir, G. Benson, and M. Farach, *Parallel two dimensional matching in logarithmic time*, in Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, 1993, pp. 79–85.

[ABG92] A. Apostolico, D. Breslauer, and Z. Galil, *Optimal parallel algorithms for periods, palindromes and squares*, in Proceedings of the 19th International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 623, Springer-Verlag, Berlin, 1992, pp. 296–307.

[AL88] A. Amir and G. Landau, *Fast parallel and serial multidimensional approximate array matching*, Theoret. Comput. Sci., 81 (1988), pp. 347–365.

[Ba78] T. J. Baker, *A technique for extending rapid exact-match string matching to arrays of more than one dimension*, SIAM J. Comput. 7 (1978), pp. 533–541.

[Bi77] R. S. Bird, *Two dimensional pattern matching*, Inform. Process. Lett., 6 (1977), pp. 168–170.

[BG90] D. Breslauer and Z. Galil, *An optimal $O(\log\log m)$ time parallel string matching algorithm*, SIAM J. Comput., 19 (1990), pp. 1051–1058.

[CC+93] R. Cole, M. Crochemore, Z. Galil, L. Gąsieniec, R. Hariharan, S. Muthukrishnan, K. Park, and W. Rytter, *Optimally fast parallel algorithms for preprocessing and pattern matching in one and two dimensions*, in Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1993, pp. 248–258.

[CG+93a] M. Crochemore, L. Gąsieniec, Z. Galil, K. Park, and W. Rytter, *Constant Time Deterministic Sample Computation and Its Applications*, manuscript, 1993.

[CG+93b] R. Cole, Z. Galil, R. Hariharan, S. Muthukrishnan, and K. Park, *Optimal Parallel Two Dimensional Witness Computation*, manuscript, 1993.

[CR91] M. Crochemore and W. Rytter, *Usefulness of the Karp-Miller-Rosenberg algorithm in parallel computations in strings and arrays*, Theoret. Comput. Sci., 88 (1991), pp. 59–62.

[CR95] M. Crochemore and W. Rytter, *On linear-time alphabet-independent 2-dimensional pattern matching*, LATIN'95, Lecture Notes in Comput. Sci. 911, Springer-Verlag, Berlin, 1995, pp. 220–229.

[FRW88] F. Fich, R. Ragde, and A. Widgerson, *Relations between concurrent-write models of parallel computation*, SIAM J. Comput., 17 (1988), pp. 606–627.

[Ga85] Z. galil, *Optimal parallel algorithms for string matching*, Inform. and Control, 67 (1985), pp. 144–157.

[Ga92] Z. Galil, *Hunting lions in the desert optimally or a constant-time optimal parallel string matching algorithm*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, ACM, New York, 1992, pp. 69–76.

[GP92] Z. Galil and K. Park, *Truly alphabet-independent two dimensional matching*, in Proceedings of the 33rd Annual IEEE Symposium on the Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 247–256.

[Ja91] J. Jaja, *Introduction to Parallel Algorithms*, Addison–Wesley, Reading, MA, 1991.

[KLP89] Z. Kedem, G. Landau, and K. Palem, *Optimal parallel suffix-prefix matching algorithm and application*, in Proceedings of the 1st Annual ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, 1989, pp. 388–398.

[KMR72] R. M. Karp, R. E. Miller, and A. L. Rosenberg, *Rapid identification of repeated patterns in strings, trees and arrays*, in Proceedings of the 4th Annual ACM Symposium on Theory of Computing, ACM, New York, 1972, pp. 125–136.

[KR87] R. Karp and M. Rabin, *Efficient randomized pattern matching algorithms*, IBM J. Res. Develop., 31 (1987), pp. 249–260.

[LS62] R. Lyndon and M. Schutzenberger, *The equation $a^M = b^N c^P$ in a free group*, Michigan Math. J., 9 (1962), pp. 289–298.

[M88] T. Mathies, *A Fast Parallel Algorithm to Determine Edit Distance*, Technical Report TR CMU-CS-88-130, Carnegie–Mellon Univ., Pittsburgh, PA, 1988.

[Ra90] P. Ragde, *The parallel simplicity of compaction and chaining*, in Proceedings of the 17th International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 443, Springer-Verlag, Berlin, 1990, pp. 744–751.

[RS62] J. B. Rosser and L. Schoenfeld, *Approximate formulas for some functions of prime numbers*, Illinois J. Math., 6 (1962), pp. 64–94.

[Vi85] U. Vishkin, *Optimal pattern matching in strings*, Inform. and Control, 67 (1985), pp. 91–113.

[Vi90] U. Vishkin, *Deterministic sampling—A new technique for fast pattern matching*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, ACM, New York, 1990, pp. 170–180.

SIAM J. COMPUT.
Vol. 27, No. 3, pp. 682–693, June 1998
© 1998 Society for Industrial and Applied Mathematics
006

IMPROVED RANDOMIZED ON-LINE ALGORITHMS FOR THE LIST UPDATE PROBLEM*

SUSANNE ALBERS†

Abstract. The best randomized on-line algorithms known so far for the list update problem achieve a competitiveness of $\sqrt{3} \approx 1.73$. In this paper we present a new family of randomized on-line algorithms that beat this competitive ratio. Our improved algorithms are called TIMESTAMP algorithms and achieve a competitiveness of $\max\{2-p, 1+p(2-p)\}$, for any real number $p \in [0,1]$. Setting $p = (3-\sqrt{5})/2$, we obtain a ϕ-competitive algorithm, where $\phi = (1+\sqrt{5})/2 \approx 1.62$ is the golden ratio. TIMESTAMP algorithms coordinate the movements of items using some information on past requests. We can reduce the required information at the expense of increasing the competitive ratio. We present a very simple version of the TIMESTAMP algorithms that is 1.68-competitive. The family of TIMESTAMP algorithms also includes a new deterministic 2-competitive on-line algorithm that is different from the MOVE-TO-FRONT rule.

Key words. linear lists, on-line algorithms, competitive analysis

AMS subject classifications. 68P05, 68P10, 68Q20, 68Q25

PII. S0097539794277858

1. Introduction.

The *list update problem* is among the first on-line problems that have been studied with respect to competitiveness. The problem consists in maintaining a set of items as an unsorted linear list. A list of n items is given. A list update algorithm is presented with a sequence of *requests* that must be served in their order of occurrence. Each request specifies an item in the list. In order to serve a request, a list update algorithm must *access* the requested item, i.e., it has to start at the front of the list and search linearly through the items until the desired item is found. Accessing the ith item in the list incurs a cost of i. Immediately after an access, the requested item may be moved at no extra cost to any position closer to the front of the list. These exchanges are called *free exchanges*. All other exchanges of two consecutive items in the list cost 1 and are called *paid exchanges*. The goal is to serve the request sequence so that the total cost is as small as possible. A list update algorithm is *on-line* if it serves every request without knowledge of any future requests.

We analyze the performance of on-line algorithms for the list update problem using *competitive analysis* [6]. In a competitive analysis, an on-line algorithm A is compared to an *optimal off-line* algorithm. An optimal off-line algorithm knows the entire request sequence in advance and can serve it with minimum cost. Given a request sequence σ, let $C_A(\sigma)$ denote the cost incurred by on-line algorithm A in serving σ and let $C_{OPT}(\sigma)$ denote the cost incurred by the optimal off-line algorithm OPT in processing σ. Then the algorithm A is called c-competitive if there is a constant a such that $C_A(\sigma) \leq c \cdot C_{OPT}(\sigma) + a$ for all request sequences σ. The *competitive factor* of A is the infimum of all c such that A is c-competitive.

Sleator and Tarjan [6] have shown that the well-known MOVE-TO-FRONT algorithm is 2-competitive. This deterministic on-line algorithm moves an item to the

*Received by the editors December 5, 1994; accepted for publication (in revised form) March 25, 1996.

http://www.siam.org/journals/sicomp/27-3/27785.html

†Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany (albers@mpi-sb.mpg.de). This research was supported in part by the ESPRIT Basic Research Actions Program of the European Union under contract no. 7141 (project ALCOM II).

front of the list each time it is requested. Karp and Raghavan [4] have observed that no deterministic on-line algorithm for the list update problem can be better than 2-competitive. Thus the MOVE-TO-FRONT algorithm achieves the best possible competitive factor. A natural question is if the competitive factor of 2 can be improved using randomization. It was shown that, against *adaptive adversaries*, no randomized on-line algorithm for the list update problem can be better than 2-competitive [1, 5]. Adaptive adversaries may see the on-line algorithm's random choices on past requests when generating a new request in a request sequence σ. On the other hand, against *oblivious adversaries*, the optimal competitive factor of randomized on-line algorithms has not yet been determined. An oblivious adversary specifies a request sequence in advance and is not allowed to see the random choices made by the on-line algorithm. A randomized on-line algorithm A is called c-competitive against any oblivious adversary if there exists a constant a such that for all request sequences σ generated by oblivious adversaries, $E[C_A(\sigma)] \leq c \cdot C_{OPT}(\sigma) + a$, where the expectation is taken over the random choices made by A. In this paper we always evaluate on-line algorithms with respect to oblivious adversaries. Irani [3] exhibited the first randomized on-line algorithm for the list update problem; the SPLIT algorithm she proposed is $\frac{31}{16}$-competitive. Reingold, Westbrook, and Sleator [5] have given a family of COUNTER and RANDOM RESET algorithms that achieve a competitive ratio of $\sqrt{3} \approx 1.73$. This has been the best upper bound known so far for randomized list update algorithms. The best lower bound known is due to Teia [7]. He shows that no randomized on-line algorithm for the list update problem can be better than 1.5-competitive.

In this paper we present improved randomized on-line algorithms for the list update problem that beat the competitive ratio of $\sqrt{3}$. Our new algorithms are called TIMESTAMP algorithms and achieve a competitiveness of $\max\{2-p, 1+p(2-p)\}$ for any real number $p \in [0,1]$. Choosing $p = (3-\sqrt{5})/2$, we obtain a ϕ-competitive algorithm, where $\phi = (1+\sqrt{5})/2 \approx 1.62$ is the golden ratio. TIMESTAMP algorithms do not always move the requested item x to the front of the list but sometimes to a position that is only a bit closer to the front. This position can be computed easily when the algorithm scans the items preceding x in the list. However, in the implementation of the algorithm, the computation of this position requires a second pass through the list after the item x has been accessed. Moreover, some information on past requests is necessary in order to determine the desired position. We can simplify the TIMESTAMP algorithms so that they need less knowledge of previous requests; this increases the competitive ratio. We present a simplified version of the TIMESTAMP algorithms that is 1.68-competitive. The family of TIMESTAMP algorithms also includes two deterministic 2-competitive on-line algorithms, one of which is the MOVE-TO-FRONT rule. The second, new algorithm is the only other deterministic on-line algorithm found so far that achieves a competitive factor of 2; Sleator and Tarjan [6] have proved that the well-known deterministic algorithms TRANSPOSE and FREQUENCY COUNT are not 2-competitive.

The list update problem as defined above is the *static* version of the problem. Each request is an access to an item. In the *dynamic* variant of the problem, insertions and deletions of items are allowed. A new item is inserted by scanning the entire list and appending the item at the end of the list. A deletion of an item is processed by searching for the item in the list and deleting it. In the following sections, when we develop and analyze randomized list update algorithms, we always consider the static version of the problem. However, the on-line algorithms we will propose can

be extended in the obvious way so that they can handle insertions and deletions, too. All theorems that we will present also hold for the dynamic list update problem.

2. TIMESTAMP algorithms. We present a new family of randomized on-line algorithms for the list update problem. The following algorithm works for any real number $p \in [0,1]$.

Algorithm TIMESTAMP(p). Given a request sequence $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$, each request $\sigma(t)$, $1 \leq t \leq m$ is processed as follows. Suppose that $\sigma(t)$ is a request to item x.

With probability p, execute Step (a).

(a) Move x to the front of the list.

With probability $1-p$, execute Step (b).

(b) If x has not been requested so far during the time interval $[1, t-1]$, then do not change the position of x in the list. Otherwise let $t' \in [1, t-1]$ be the time at which x was requested most recently and serve the request $\sigma(t)$ as follows. Let $v_x(t)$ be the item closest to the front of the list that precedes x in the list and

(i) that was not requested during the interval $[t', t-1]$

or

(ii) that was requested exactly once during $[t', t-1]$ and the corresponding request was served using Step (b) of the algorithm.

If there is no such item, then let $v_x(t) = x$.

Insert x immediately before $v_x(t)$.

THEOREM 1. *For any real number $p \in [0,1]$, TIMESTAMP(p) is c-competitive, where $c = \max\{2-p,\ 1+p(2-p)\}$.*

COROLLARY 1. *TIMESTAMP($\frac{3-\sqrt{5}}{2}$) is ϕ-competitive, where $\phi = \frac{1}{2}(1+\sqrt{5})$ is the golden ratio.*

An interesting feature of the TIMESTAMP(p) algorithm is that at a request to item x, all items y satisfying condition (i) or (ii) in Step (b) are stored consecutively in front of x. In other words, all items stored between x and $v_x(t)$ satisfy condition (i) or (ii); we will prove this later in Lemma 2.

We assume that the algorithm TIMESTAMP(p) maintains a time stamp $ST(y)$ for each item y in the list. While a request sequence in served, $ST(y)$ always stores the time of the most recent request to y. When there is a request to item x, TIMESTAMP(p) can easily determine the first item in the list that satisfies condition (i) in Step (b). The algorithm just has to find the first item y with $ST(y) < ST(x)$. Finding the first item in the list that satisfies condition (ii) in Step (b) requires more information on previous requests. We can simplify the TIMESTAMP(p) algorithm at the expense of increasing the competitive ratio. Suppose that we drop condition (ii) in Step (b). Then $v_x(t)$ is simply the item closest to the front of the list that precedes x and has not been requested during the interval $[t', t-1]$. We can show the following performance.

THEOREM 2. *If condition* (ii) *is dropped in Step* (b) *of TIMESTAMP(p), then the resulting algorithm is c-competitive, where $c = \max\{2-p, 1+p(2-p), 2-\frac{3}{2}p+2p^2-\frac{1}{2}p^3\}$. Setting $p = \frac{1}{2}(5-\sqrt{17})$, we obtain a competitive ratio of $\frac{3}{2}(\sqrt{17}-3) \approx 1.68$.*

We will prove this theorem after we have proven Theorem 1.

Although the item $v_x(t)$ specified in TIMESTAMP(p) can be computed easily, in a real implementation of the algorithm we need a second pass through the list in order to actually locate $v_x(t)$. Note that a number of on-line algorithms that have been proposed in the literature, such as the algorithm FREQUENCY COUNT, also

require such a second pass after each access to an item. There is an alternative formulation of TIMESTAMP(p) that does not need a second pass. In this alternative formulation, we maintain a pointer $ptr(x)$ for each item x in the list; $ptr(x)$ either points to x or to an item preceding x in the list. At a request to item x, x is inserted at the front of the list or immediately before $ptr(x)$. The element specified by $ptr(x)$ always corresponds to $v_x(t)$. The drawback of this alternative formulation is that we need time to update the pointers. Essentially, at a request to item x, the pointers of all elements y with $ptr(y) = x$ must be set to the successor of x in the list, and $ptr(x)$ must be set to the front of the list. We prefer the formulation of the TIMESTAMP(p) algorithm given above because it explicitly describes the properties of the position $v_x(t)$ in front of which an item x is to be inserted.

TIMESTAMP(p) describes two deterministic algorithms. Setting $p = 1$ we obtain the MOVE-TO-FRONT algorithm. Theorems 1 and 2 confirm the well-known fact that the MOVE-TO-FRONT rule is 2-competitive. On the other hand, assume $p = 0$ and consider the simplified version of the TIMESTAMP algorithm in which condition (ii) of Step (b) is dropped. The resulting deterministic algorithm always inserts the requested item x immediately before the first item in the list that has not been requested since the last request to x. Theorem 2 implies that this deterministic strategy is 2-competitive.

We now proceed with the proof of Theorem 1. Consider a fixed $p \in [0, 1]$. Let $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$ be an arbitrary request sequence consisting of m requests and let $\sigma(t)$ denote the request at time t, $1 \leq t \leq m$. We first present two lemmata that describe the relative positions of items in the list while TIMESTAMP(p) serves a request sequence.

LEMMA 1. *Let x and y, $x \neq y$ be two items. Suppose that x is requested at time t' and at time t, $t' < t$, and that y is not requested during the interval $[t', t]$. Then, immediately after the service of $\sigma(t)$, x precedes y in TIMESTAMP(p)'s list and this relation does not change before the next request to y.*

Proof. If TIMESTAMP(p) executes Step (a) when serving $\sigma(t)$, then x is moved to the front of the list and must precede y in the list. If TIMESTAMP(p) executes Step (b) when serving $\sigma(t)$, then condition (i) in Step (b) of the algorithm ensures that x is inserted at some position in front of item y because y is not requested during $[t', t]$. In any case, x precedes y in TIMESTAMP(p)'s list immediately after the service of $\sigma(t)$. Since y is only moved when it is requested, the relative position of x and y cannot change before the next request to y. □

LEMMA 2. *Let t be a time in $[1, m]$. If the item $x = \sigma(t)$ was requested at least once in $[1, t-1]$, then the following two statements hold. Let t', $t' < t$ denote the time at which x was requested most recently.*

(a) *If item y, $y \neq x$ was requested at least twice during $[t', t-1]$, then y precedes item $v_x(t)$ in TIMESTAMP(p)'s list at time t.*
(b) *If item y, $y \neq x$ was requested exactly once during $[t', t-1]$ and the corresponding request was served using Step* (a) *of TIMESTAMP(p), then y precedes item $v_x(t)$ in TIMESTAMP(p)'s list at time t.*

Proof. Suppose that there is a time in $[1, m]$ at which Lemma 2 does not hold. Then let $t_0 \in [1, m]$ be the earliest point of time at which the lemma is violated. Furthermore, let t_0', $t_0' < t_0$ be the time at which item $x = \sigma(t_0)$ was requested most recently, and let $z = v_x(t_0)$.

First we examine the case that statement (a) of the lemma does not hold. Thus, there exists an item y, $y \neq x$ that is requested at least twice in $[t_0', t_0 - 1]$ and that does not precede z at time t_0. Let t_y be the time of the last request to y in $[t_0', t_0 - 1]$. We show that after the service of $\sigma(t_y)$, item y precedes z in TIMESTAMP(p)'s list. If

$\sigma(t_y)$ is served using Step (a) of the algorithm, then there is nothing to show. Suppose that $\sigma(t_y)$ is served using Step (b) of TIMESTAMP(p). By the definition of $v_x(t_0)$, item z is requested at most once in $[t_0', t_0 - 1]$, and such a request is served using Step (b) of the algorithm. This implies that when TIMESTAMP(p) serves $\sigma(t_y)$, z cannot precede $v_y(t_y)$ (due to conditions (i) and (ii) in Step (b) of the algorithm). Hence y is inserted at some position in front of z. We conclude that y must precede z after the service of $\sigma(t_y)$. Since, by assumption, z precedes y at time t_0, item z must be requested at some time $t_z \in [t_y + 1, t_0 - 1]$ and z must be inserted at some position in front of y when $\sigma(t_z)$ is served. This implies that y cannot precede $v_z(t_z)$ at time t_z because $\sigma(t_z)$ is served using Step (b) of the algorithm. Note that at time t_z, y was requested at least twice since the last request to z. Hence statement (a) of the lemma does not hold at time t_z, and we have a contradiction to the minimality of t_0.

Now assume that statement (b) of the lemma is violated. Let y, $y \neq x$ be an item for which statement (b) does not hold, and let $t_y \in [t_0', t_0 - 1]$ be the time at which y is requested. By assumption, y does not precede $z = v_x(t_0)$ at time t_0. Since $\sigma(t_y)$ is served using Step (a) of the algorithm, y precedes z after the service of $\sigma(t_y)$. Using the same arguments as above, we can derive a contradiction to the choice of t_0. □

In the following we will evaluate TIMESTAMP(p)'s and OPT's cost on request sequence σ. Let $C_{TS}(\sigma)$ be the cost incurred by TIMESTAMP(p) in serving σ. We will show that

$$E[C_{TS}(\sigma)] \leq c \cdot C_{OPT}(\sigma), \tag{2.1}$$

where $c = \max\{2 - p, 1 + p(2 - p)\}$. This proves Theorem 1. Here we assume, without loss of generality, that TIMESTAMP(p) and OPT start with the same initial list. In the following, when analyzing on-line and off-line cost, we will always use the $(i-1)$-*cost measure*, i.e., we assume that an access to the ith item in the list incurs a cost of $i - 1$ rather than i. Obviously, an on-line algorithm that is c-competitive in the $(i-1)$-cost measure is also c-competitive in the i-cost measure.

We need some notation. Let L be the set of items in the list. For any $t \in [1, m]$ and any item $x \in L$, let $C_{TS}(t, x)$ be the cost incurred by item x when TIMESTAMP(p) serves $\sigma(t)$. More precisely, $C_{TS}(t, x) = 1$ if at time t, item x precedes the item requested by $\sigma(t)$ in TIMESTAMP(p)'s list; otherwise $C_{TS}(t, x) = 0$. We have

$$\begin{aligned} E[C_{TS}(\sigma)] &= E\left[\sum_{t\in[1,m]} \sum_{x\in L} C_{TS}(t,x)\right] \\ &= E\left[\sum_{x\in L} \sum_{t\in[1,m]} C_{TS}(t,x)\right] \\ &= E\left[\sum_{x\in L} \sum_{y\in L} \sum_{\substack{t\in[1,m] \\ \sigma(t)=y}} C_{TS}(t,x)\right]. \end{aligned}$$

Let $\{x, y\}$ be an unordered pair of items x and y with $x \neq y$. Every pair $\{x, y\}$ contributes two terms in the last line of the above equation, namely,

$$\sum_{\substack{t\in[1,m] \\ \sigma(t)=x}} C_{TS}(t,y) \quad \text{and} \quad \sum_{\substack{t\in[1,m] \\ \sigma(t)=y}} C_{TS}(t,x).$$

Thus,

$$E[C_{TS}(\sigma)] = E\left[\sum_{\substack{\{x,y\}\\ x\neq y}}\left(\sum_{\substack{t\in[1,m]\\ \sigma(t)=x}} C_{TS}(t,y) + \sum_{\substack{t\in[1,m]\\ \sigma(t)=y}} C_{TS}(t,x)\right)\right]. \tag{2.2}$$

The cost incurred by OPT can be written in a similar way, i.e.,

$$C_{OPT}(\sigma) = \sum_{\substack{\{x,y\}\\ x\neq y}}\left(\sum_{\substack{t\in[1,m]\\ \sigma(t)=x}} C_{OPT}(t,y) + \sum_{\substack{t\in[1,m]\\ \sigma(t)=y}} C_{OPT}(t,x) \;+\; p(x,y)\right). \tag{2.3}$$

Here $C_{OPT}(t,y)$ and $C_{OPT}(t,x)$ denote the costs incurred by items y and x when OPT serves $\sigma(t)$. For any unordered pair $\{x,y\}$ of items $x \neq y$, $p(x,y)$ denotes the total number of paid exchanges that OPT incurs in moving x in front of y or y in front of x.

Now, for any pair $\{x,y\}$ of items with $x \neq y$, let σ_{xy} be the request sequence that is obtained from σ if we delete all requests in σ that are neither to x nor to y. Let $E[C_{TS}(\sigma_{xy})]$ be the expected cost incurred by TIMESTAMP(p) if it serves σ_{xy} on a list consisting of x and y only. Lemma 2 implies that if TIMESTAMP(p) serves a request $\sigma(t)$ in σ using Step (b), then the requested item $x = \sigma(t)$ never passes an item that was requested at least twice since the last request to x, or an item that was requested exactly once and the corresponding request was served using Step (a) of the algorithm. Thus, for any pair $\{x,y\}$ of items, the following statement holds. If TIMESTAMP(p) serves σ on the entire list, then the relative position of x and y changes in the same way as if TIMESTAMP(p) is run on the two item list consisting of x and y with request sequence σ_{xy}. Therefore, we have

$$E[C_{TS}(\sigma_{xy})] = E\left[\sum_{\substack{t\in[1,m]\\ \sigma(t)=x}} C_{TS}(t,y) + \sum_{\substack{t\in[1,m]\\ \sigma(t)=y}} C_{TS}(t,x)\right]$$

and, by equation (2.2),

$$E[C_{TS}(\sigma)] = \sum_{\substack{\{x,y\}\\ x\neq y}} E[C_{TS}(\sigma_{xy})].$$

Let $C_{OPT}(\sigma_{xy})$ be the cost incurred by OPT if it serves σ_{xy} on the list consisting of x and y only. In this two item list, OPT can always arrange x and y optimally, which might not be possible when OPT serves σ on the entire list. Hence

$$C_{OPT}(\sigma_{xy}) \leq \sum_{\substack{t\in[1,m]\\ \sigma(t)=x}} C_{OPT}(t,y) + \sum_{\substack{t\in[1,m]\\ \sigma(t)=y}} C_{OPT}(t,x) \;+\; p(x,y),$$

and equation (2.3) implies

$$C_{OPT}(\sigma) \geq \sum_{\substack{\{x,y\}\\ x\neq y}} C_{OPT}(\sigma_{xy}).$$

This method of analyzing cost by considering pairs of items was also used in [2, 3]. In the following we show that for any pair of items $\{x, y\}$ with $x \neq y$,

$$E[C_{TS}(\sigma_{xy})] \leq c \cdot C_{OPT}(\sigma_{xy}), \tag{2.4}$$

where $c = \max\{2 - p, 1 + p(2 - p)\}$. This proves inequality (2.1).

Consider an arbitrary pair $\{x, y\}$ with $x \neq y$. Let $\sigma_{xy} = \sigma(t_1), \sigma(t_2), \ldots, \sigma(t_k)$ for some nonnegative integer k. For $i = 1, 2, \ldots, k$, let u_i be the item requested by $\sigma(t_i)$. Lemma 1 suggests partitioning σ_{xy} into phases $P(1), P(2), \ldots, P(l)$ for some l so that the following condition holds. If phase $P(j)$, $1 \leq j \leq l$ starts at time t_{b_j}, then it ends at time t_{e_j}, where

$$e_j = \min\{i > b_j | u_{i-1} = u_i \text{ and } u_i \neq u_{i+1}\}.$$

In other words, a phase ends when, for the first time, there have been two consecutive requests to the same item and the next request is different. The phases we obtain can be classified as follows.

Type 1:	(a) $P(j) = x^h$	or	(b) $P(j) = y^h$	for some $h \geq 2$.
Type 2:	(a) $P(j) = xy^h$	or	(b) $P(j) = yx^h$	for some $h \geq 2$.
Type 3:	(a) $P(j) = (xy)^{h_1}x^{h_2}$	or	(b) $P(j) = (yx)^{h_1}y^{h_2}$	for some $h_1 \geq 1, h_2 \geq 2$.
Type 4:	(a) $P(j) = (xy)^{h_1}y^{h_2}$	or	(b) $P(j) = (yx)^{h_1}x^{h_2}$	for some $h_1 \geq 2, h_2 \geq 1$.

We may assume, without loss of generality, that the item first requested in a phase $P(j)$, $1 \leq j \leq l$ is behind the other item of the pair $\{x, y\}$ in the two item lists maintained by TIMESTAMP(p) and OPT. This is easy to see for phase $P(1)$. If the first item in $P(1)$, say x, precedes y in the initial list, then we can simply omit the first requests to x in σ_{xy} until we obtain the first request to y. This does not change the cost incurred by TIMESTAMP(p) and OPT in $P(1)$ because we assume that TIMESTAMP(p) and OPT start with the same initial list. Now consider a phase $P(j)$, $2 \leq j \leq l$. The item first requested in $P(j)$ differs from the item requested by the two previous requests. Lemma 1 immediately implies that the first item in $P(j)$ is behind the other item of the pair $\{x, y\}$ in the list maintained by TIMESTAMP(p). Consider OPT's movements when it serves the request sequence σ_{xy} on the two item list. We may assume, without loss of generality, that whenever there are two consecutive requests to the same item, OPT moves that item to the front of the list, provided that it has not been there yet. This implies, without loss of generality, that immediately before the first request in a phase $P(j)$, the item requested first in the phase is also behind the other item of the pair $\{x, y\}$ in OPT's two item list.

In the following we evaluate the expected cost incurred by TIMESTAMP(p) and the cost incurred by OPT in each phase of σ_{xy}. For each $j = 1, 2, \ldots, l$, the expected cost incurred by TIMESTAMP(p) in phase $P(j)$ of σ_{xy} is

$$E[C_{TS}(P(j))] = E\left[\sum_{i=b_j}^{e_j}(C_{TS}(t_i, y) + C_{TS}(t_i, x))\right].$$

Similarly, for $j = 1, 2, \ldots, l$, let $C_{OPT}(P(j))$ be the cost incurred by OPT when it serves phase $P(j)$ of σ_{xy}. We will prove the following lemmata.

LEMMA 3. *If $P(j)$ has type 1, then $E[C_{TS}(P(j))] \leq (2 - p)C_{OPT}(P(j))$.*

LEMMA 4. *If $P(j)$ has type 2, then $E[C_{TS}(P(j))] \leq (1 + p(2 - p))C_{OPT}(P(j))$.*

LEMMA 5. *If $P(j)$ has type 3 or 4, then $E[C_{TS}(P(j))] \leq \max\{2-p, 1+p(2-p)\}C_{OPT}(P(j))$.*

Before we prove the lemmata we finish the proof of inequality (2.4). Lemmata 3–5 imply that

$$\begin{aligned} E[C_{TS}(\sigma_{xy})] &= E\left[\sum_{j=1}^{l} C_{TS}(P(j))\right] \\ &\leq \max\{2-p, 1+p(2-p)\}\sum_{j=1}^{l} C_{OPT}(P(j)) \\ &= \max\{2-p, 1+p(2-p)\}C_{OPT}(\sigma_{xy}), \end{aligned}$$

and inequality (2.4) is proved.

We have classified phases $P(1), P(2), \ldots, P(l)$ into four types. For each type, subtypes (a) and (b) are symmetric to each other. In the following, we will always assume, without loss of generality, that the considered phase has subtype (a). Let $P(j)$ be an arbitrary phase. By the above discussion we know that immediately before the first request in $P(j)$, x is behind y in the two item lists maintained by TIMESTAMP(p) and OPT. Thus $E[C_{TS}(t_{b_j}, y)] = C_{OPT}(t_{b_j}, y) = 1$, for $j = 1, 2, \ldots, l$.

Claim 1 below will be useful when proving Lemmata 3–5. We will present a proof of this claim later.

CLAIM 1. *After the service of the first request $\sigma(t_{b_j})$ in a phase $P(j)$, $1 \leq j \leq l$, item $x = \sigma(t_{b_j})$ precedes item y if and only if TIMESTAMP(p) serves $\sigma(t_{b_j})$ using Step* (a) *of the algorithm.*

Proof of Lemma 3. We have $C_{TS}(P(j)) = \sum_{i=b_j}^{e_j} C_{TS}(t_i, y)$. By Lemma 1, x precedes y in TIMESTAMP(p)'s list after the service of $\sigma(t_{b_j+1})$. Hence, y cannot cause a cost at the third and all remaining requests to x in $P(j)$. We obtain

$$\begin{aligned} E[C_{TS}(P(j))] &= E[C_{TS}(t_{b_j}, y)] + E[C_{TS}(t_{b_j+1}, y)] \\ &= 1 + E[C_{TS}(t_{b_j+1}, y)]. \end{aligned}$$

$E[C_{TS}(t_{b_j+1}, y)]$ is the probability that item $x = u_{b_j+1}$ is behind item y in TIMESTAMP(p)'s list when $\sigma(t_{b_j+1})$ is served. Applying Claim 1 we infer that x is behind y if and only if TIMESTAMP(p) serves $\sigma(t_{b_j})$ using Step (b), which happens with probability $1-p$. We conclude $E[C_{TS}(t_{b_j+1}, y)] = 1-p$ and $E[C_{TS}(P(j))] = 2-p$. Since $C_{OPT}(P(j)) = 1$, the lemma follows. □

Proof of Lemma 4. TIMESTAMP(p)'s cost in phase $P(j)$ is $C_{TS}(P(j)) = C_{TS}(t_{b_j}, y) + \sum_{i=b_j+1}^{e_j} C_{TS}(t_i, x)$. Lemma 1 implies that x cannot cause a cost at the third and all remaining requests to y in $P(j)$. Hence

$$\begin{aligned} E[C_{TS}(P(j))] &= E[C_{TS}(t_{b_j}, y)] + E[C_{TS}(t_{b_j+1}, x)] + E[C_{TS}(t_{b_j+2}, x)] \\ &= 1 + E[C_{TS}(t_{b_j+1}, x)] + E[C_{TS}(t_{b_j+2}, x)]. \end{aligned}$$

We have $C_{TS}(t_{b_j+1}, x) = 1$ if the TIMESTAMP(p) algorithm moves x in front of y when serving $\sigma(t_{b_j})$. By Claim 1, this happens if TIMESTAMP(p) processes $\sigma(t_{b_j})$ using Step (a). Therefore, $E[C_{TS}(t_{b_j+1}, x)] = p$. We analyze $E[C_{TS}(t_{b_j+2}, x)]$. We have $C_{TS}(t_{b_j+2}, x) = 1$ if TIMESTAMP(p) moves x in front of y when serving $\sigma(t_{b_j})$ and does not move y in front of x when serving $\sigma(t_{b_j+1})$. Claim 1 implies that with probability p, TIMESTAMP(p) moves x in front of y when serving $\sigma(t_{b_j})$. Item y

can only stay behind x in TIMESTAMP(p)'s list if $\sigma(t_{b_j+1})$ is served using Step (b), which happens with probability $1-p$. Thus, the expected cost on $\sigma(t_{b_j+2})$ is at most $p(1-p)$. We obtain $E[C_{TS}(P(j))] \leq 1+p+p(1-p) = 1+p(2-p)$, and the lemma follows because $C_{OPT}(P(j)) = 1$. □

Proof of Lemma 5. We know that $C_{TS}(t_{b_j}, y) = 1$ and, using Claim 1, $E[C_{TS}(t_{b_j+1}, x)] = p$. First we assume that $P(j)$ has type 3. Then $P(j)$ consists of a head of $2h_1$ alternating requests to x and y and of a tail of h_2 requests to x. Lemma 1 ensures that in the tail of requests to x, item x must precede y in TIMESTAMP(p)'s list after the service of the second request to x. Hence y cannot cause a cost at any of the remaining requests in $P(j)$, i.e., $C_{TS}(t_i, y) = 0$ for $i = b_j + 2(h_1+1), \ldots, e_j$. Thus

$$C_{TS}(P(j)) = 1 + p + \sum_{i=b_j+2}^{b_j+2h_1} C_{TS}(t_i, u_{i-1}) + C_{TS}(t_{b_j+2h_1+1}, y).$$

For the analysis of the cost incurred at the alternating requests to x and y we need the following claim that we will prove later.

Claim 2. *Let $\sigma(t_{i-3})\sigma(t_{i-2})\sigma(t_{i-1}) = xyx$ or $\sigma(t_{i-3})\sigma(t_{i-2})\sigma(t_{i-1}) = yxy$ be three consecutive requests in σ_{xy} with $4 \leq i \leq k$. Then, after the service of $\sigma(t_{i-1})$, with probability $1-p+p^2$, u_{i-1} precedes u_{i-2} in the list maintained by TIMESTAMP(p).*

If we have a phase $P(j)$ with $j \geq 2$, then Claim 2 implies that, for $i = b_j + 2, \ldots, b_j + 2h_1$, immediately before the request to $\sigma(t_i)$, item u_{i-1} precedes $u_i = u_{i-2}$ with probability $1-p+p^2$. Therefore, $E[C_{TS}(t_i, u_{i-1})] = 1-p+p^2$ for $i = b_j + 2, \ldots, b_j + 2h_1$. We remark that we may apply Claim 2 for $i = b_j + 2$ because $u_{b_j-1} \neq u_{b_j}$. For phase $P(1)$, Claim 2 gives $E[C_{TS}(t_i, u_{i-1})] = 1-p+p^2$ for $i = b_1+3, \ldots, b_1+2h_1$. However, it is easy to show that for request $\sigma(t_{b_1+2})$, $E[C_{TS}(t_{b_1+2}, u_{b_1+1})] = 1-p+p^2$. We now evaluate the cost $C_{TS}(t_{b_j+2h_1+1}, y)$. Claim 2 gives that immediately before the request $\sigma(t_{b_j+2h_1+1})$, item x precedes item y with probability $1-p+p^2$ in TIMESTAMP(p)'s list. Hence $E[C_{TS}(t_{b_j+2h_1+1}, y)] = 1-(1-p+p^2) = p-p^2$. We conclude that

$$\begin{aligned} E[C_{TS}(P(j))] &= 1 + p + (2h_1 - 1)(1-p+p^2) + p - p^2 \\ &= 2(h_1+1)(1-p+p^2) - 2 + 5p - 4p^2 \\ &\leq 2(h_1+1)(1-p+p^2) \end{aligned}$$

for all $p \in [0,1]$. Thus

$$\begin{aligned} E[C_{TS}(P(j))] &\leq 2(h_1+1)(1-p+p^2) \\ &= (h_1+1)(2-p) + (h_1+1)(-p+2p^2) \\ &\leq (h_1+1)(2-p) \end{aligned}$$

for all $p \leq \frac{1}{2}$. Moreover,

$$\begin{aligned} E[C_{TS}(P(j))] &\leq 2(h_1+1)(1-p+p^2) \\ &= (h_1+1)(1+p(2-p)) + (h_1+1)(1-4p+3p^2) \\ &\leq (h_1+1)(1+p(2-p)) \end{aligned}$$

for all $p \geq \frac{1}{3}$. Since $C_{OPT}(P(j)) = h_1 + 1$, we obtain

$$E[C_{TS}(P(j))] \leq \max\{2-p, 1+p(2-p)\} C_{OPT}(P(j)).$$

The analysis for a phase $P(j)$ having type 4 is very similar. By Lemma 1, $C_{TS}(t_i, x) = 0$ for $i = b_j + 2h_1 + 1, \ldots, e_j$. Hence

$$C_{TS}(P(j)) = 1 + p + \sum_{i=b_j+2}^{b_j+2h_1-1} C_{TS}(t_i, u_{i-1}) + C_{TS}(t_{b_j+2h_1}, x).$$

Applying Claim 2 we obtain $E[C_{TS}(t_i, u_{i-1})] = 1-p+p^2$, for $i = b_j+2, \ldots, b_j+2h_1-1$, and $E[C_{TS}(t_{b_j+2h_1}, x)] = p - p^2$. Thus

$$\begin{aligned} E[C_{TS}(P(j))] &= 1 + p + 2(h_1 - 1)(1 - p + p^2) + p - p^2 \\ &= h_1(2 - p) + h_1(-p + 2p^2) - 1 + 4p - 3p^2 \\ &\leq h_1(2 - p) \end{aligned}$$

for all $p \leq \frac{1}{3}$. Furthermore, we can show

$$\begin{aligned} E[C_{TS}(P(j))] &= h_1(1 + p(2 - p)) + (h_1 - 1)(1 - 4p + 3p^2) \\ &\leq h_1(1 + p(2 - p)) \end{aligned}$$

for all $p \geq \frac{1}{3}$. We have $C_{OPT}(P(j)) = h_1$ and therefore,

$$E[C_{TS}(P(j))] \leq \max\{2 - p, 1 + p(2 - p)\} C_{OPT}(P(j)). \qquad \square$$

Proof of Claim 1. We know that item x is behind y in TIMESTAMP(p)'s list before the service of $\sigma(t_{b_j})$. If TIMESTAMP(p) executes Step (a) when serving $\sigma(t_{b_j})$, then x is moved to the front of the list and must precede y. On the other hand, suppose that $\sigma(t_{b_j})$ is served using Step (b). If x was not requested during $[1, t_{b_j} - 1]$, then the position of x remains unchanged and x stays behind y in the list. If x was requested at least once in $[1, t_{b_j} - 1]$, then Lemma 2a implies that y precedes $v_x(t_{b_j})$ at time t_{b_j}. Again, x cannot be moved in front of y during the service of $\sigma(t_{b_j})$. $\square$

Proof of Claim 2. We analyze the sequence $\sigma(t_{i-3})\sigma(t_{i-2})\sigma(t_{i-1}) = xyx$. The case $\sigma(t_{i-3})\sigma(t_{i-2})\sigma(t_{i-1}) = yxy$ is symmetric. We have to compute the probability that x precedes y in TIMESTAMP(p)'s list after the service of $\sigma(t_{i-1})$. If TIMESTAMP(p) serves $\sigma(t_{i-1})$ using Step (a) of the algorithm, then x is moved to the front of the list and precedes y. Now assume that $\sigma(t_{i-1})$ is served using Step (b) of TIMESTAMP(p). If $\sigma(t_{i-2})$ was processed using Step (b) of the algorithm, then condition (ii) in Step (b) ensures that x is inserted at some position in front of y when TIMESTAMP(p) serves $\sigma(t_{i-1})$. On the other hand, if $\sigma(t_{i-2})$ was processed using Step (a) of the algorithm, then Lemma 2b implies that y precedes $v_x(t_{i-1})$ and x is inserted behind y when TIMESTAMP(p) serves $\sigma(t_{i-1})$.

We conclude that item x precedes item y in TIMESTAMP(p)'s list after the service of $\sigma(t_{i-1})$ if and only if one of the following events occurs. (A) TIMESTAMP(p) serves $\sigma(t_{i-1})$ using Step (a); (B) TIMESTAMP(p) serves $\sigma(t_{i-2})$ and $\sigma(t_{i-1})$ using Step (b). Event (A) occurs with probability p whereas event (B) occurs with probability $(1 - p)^2$. Thus, with probability $p + (1 - p)^2 = 1 - p + p^2$, x precedes y in TIMESTAMP(p)'s list after the service of $\sigma(t_{i-1})$. $\square$

In the above analysis we assume that the last phase $P(l)$ is a full phase of one of the phase types 1–4. It is easy to see that Lemmata 3–5 also hold for a phase $P(l)$ that is a prefix of one of the phase types. The proof of Theorem 1 is complete.

Next we analyze the performance of the simplified TIMESTAMP(p) algorithm.

Proof of Theorem 2. The proof has the same structure as the proof of Theorem 1. The statements of Lemmas 1 and 2 can be shown in the same way as before. The proofs of the lemmata become simpler, though, because we do not have to consider items that satisfy condition (ii) in Step (b) of the TIMESTAMP(p) algorithm. We compare the on-line and off-line cost for each pair $\{x, y\}$ of items with $x \neq y$ and prove

$$E\left[\sum_{\substack{t\in[1,m]\\ \sigma(t)=x}} C_{TS}(t,y) + \sum_{\substack{t\in[1,m]\\ \sigma(t)=y}} C_{TS}(t,x)\right] \leq c \cdot C_{OPT}(\sigma_{xy}), \tag{2.5}$$

where $c = \max\{2-p, 1+p(2-p), 2-\frac{3}{2}p+p^2-\frac{1}{2}p^3\}$. We partition the request sequence σ_{xy} in the same way as before. Lemmata 3 and 4 as well as their proofs (including Claim 1) remain the same. In the proof of Claim 2, only a weaker statement can be shown. We know that item u_{i-1} precedes item u_{i-2} after the service of $\sigma(t_{i-1})$ if the simplified TIMESTAMP(p) algorithm serves $\sigma(t_{i-1})$ using Step (a). Also, u_{i-1} cannot precede u_{i-2} if the simplified algorithm serves $\sigma(t_{i-2})$ using Step (a) and $\sigma(t_{i-1})$ using Step (b). Unfortunately, we do not have information about the relative position of u_{i-2} and u_{i-1} if the simplified algorithm processes $\sigma(t_{i-2})$ and $\sigma(t_{i-1})$ using Step (b). Therefore, Claim 2 changes to the following statement.

CLAIM 3. *Let $\sigma(t_{i-3})\sigma(t_{i-2})\sigma(t_{i-1}) = xyx$ or $\sigma(t_{i-3})\sigma(t_{i-2})\sigma(t_{i-1}) = yxy$ be three consecutive requests in σ_{xy} with $4 \leq i \leq k$. Then, after the service of $\sigma(t_{i-1})$, with probability at most $1-p+p^2$, u_{i-1} precedes u_{i-2} in the list maintained by the simplified TIMESTAMP(p) algorithm.*

For a proof of a statement corresponding to Lemma 5, we first consider a phase $P(j)$ having type 3. Again,

$$C_{TS}(P(j)) = 1 + p + \sum_{i=b_j+2}^{b_j+2h_1} C_{TS}(t_i, u_{i-1}) + C_{TS}(t_{b_j+2h_1+1}, y).$$

We examine $C_{TS}(t_{b_j+2h_1+1}, y)$. We have $C_{TS}(t_{b_j+2h_1+1}, y) = 1$ if $C_{TS}(t_{b_j+2h_1}, y) = 1$ and item $x = u_{b_j+2h_1}$ is not moved in front of y at time $t_{b_j+2h_1}$. Item x can only stay behind y if $\sigma(t_{b_j+2h_1})$ is served using Step (b) of the algorithm. The event that the simplified TIMESTAMP algorithm serves $\sigma(t_{b_j+2h_1})$ using Step (b) is independent of the event that $C_{TS}(t_{b_j+2h_1}, y) = 1$. Hence $E[C_{TS}(t_{b_j+2h_1+1}, y)] \leq (1-p+p^2)(1-p)$. This implies

$$E[C_{TS}(P(j))] \leq 1 + p + (2h_1 - 1)(1-p+p^2) + (1-p)(1-p+p^2).$$

Using the same techniques as in the proof of Lemma 5, we can show

$$E[C_{TS}(P(j))] \leq \max\{2-p, 1+p(2-p)\} \cdot C_{OPT}(P(j)).$$

Now consider a phase $P(j)$ having type 4. We have

$$E[C_{TS}(P(j))] \leq 1 + p + 2(h_1 - 1)(1-p+p^2) + (1-p)(1-p+p^2).$$

Hence

$$\begin{aligned} E[C_{TS}(P(j))] &\leq 2h_1(1-p+p^2) - (1+p)(-p+p^2) \\ &= h_1(2-2p+2p^2) + p(1-p^2). \end{aligned}$$

Since $h_1 \geq 2$,

$$E[C_{TS}(P(j))] \leq h_1 \left(2 - 2p + 2p^2 + \frac{p}{2}(1-p^2)\right)$$
$$= h_1 \left(2 - \frac{3}{2}p + 2p^2 - \frac{1}{2}p^3\right).$$

The statement corresponding to Lemma 5 is as follows.

LEMMA 6. *If $P(j)$ has type 3 or 4, then $E[C_{TS}(P(j))] \leq c \cdot C_{OPT}(P(j))$, where $c = \max\{2-p,\ 1+p(2-p), 2 - \frac{3}{2}p + 2p^2 - \frac{1}{2}p^3\}$.*

Lemma 6 and the statements of Lemmata 3 and 4 imply inequality (2.5). The proof of Theorem 2 is complete. □

We conclude with some remarks. TIMESTAMP algorithms use $\Theta(m)$ random bits on a request sequence of length m. We can modify the original version of TIMESTAMP(p) so that it uses only $O(n)$ random bits during an initialization phase and runs completely deterministically thereafter. The competitive ratio is still $c = \max\{2-p, 1+p(2-p)\}$. The idea is to have two different types of items, item type (a) and item type (b). Initially, we decide for each item in the list which type it should have. With probability p an item has type (a), and with probability $1-p$ it has type (b); the initializations are done independently. When a request sequence is served, each request to a type (a) item is served using Step (a) of the algorithm and every request to a type (b) item is served using Step (b). This technique cannot be applied in the simplified TIMESTAMP algorithm in which condition (ii) in Step (b) is dropped. Our analysis of the simplified TIMESTAMP algorithm makes use of the fact that the decision of whether a given request is processed using Step (a) or (b) does not depend on previous requests (see the analysis after Claim 3). In the simplified TIMESTAMP algorithm we can reduce the number of random bits using a technique presented by Reingold, Westbrook, and Sleator [5]. For each item in the list we maintain a mod i counter, where i is a positive integer. These counters are initialized independently and uniformly at random to a value in $\{0, 1, \ldots, i-1\}$. Furthermore, we choose a nonempty subset I of $\{0, 1, \ldots, i-1\}$. At a request to item x, the TIMESTAMP algorithm first decrements x's counter by 1. If the counter is I, the algorithm serves the request using Step (a); otherwise it executes Step (b). Choosing i and I appropriately, we can achieve a competitive ratio of $c + \epsilon$ for any $\epsilon > 0$; here $c = \max\{2-p, 1+p(2-p), 2 - \frac{3}{2}p + 2p^2 - \frac{1}{2}p^3\}$.

Acknowledgment. The author thanks Rudolf Fleischer and Stefan Schirra for reading an earlier version of this paper.

REFERENCES

[1] S. BEN-DAVID, A. BORODIN, R.M. KARP, G. TARDOS, AND A. WIGDERSON, *On the power of randomization in on-line algorithms*, Algorithmica, 11 (1994), pp. 2–14.

[2] J.L. BENTLEY AND C.C. MCGEOCH, *Amortized analyses of self-organizing sequential search heuristics*, Comm. ACM, 28 (1985), pp. 404–411.

[3] S. IRANI, *Two results on the list update problem*, Inform. Process. Lett., 38 (1991), pp. 301–306.

[4] R. KARP AND P. RAGHAVAN, From a personal communication cited in [5].

[5] N. REINGOLD, J. WESTBROOK, AND D.D. SLEATOR, *Randomized competitive algorithms for the list update problem*, Algorithmica, 11 (1994), pp. 15–32.

[6] D.D. SLEATOR AND R.E. TARJAN, *Amortized efficiency of list update and paging rules*, Comm. ACM, 28 (1985), pp. 202–208.

[7] B. TEIA, *A lower bound for randomized list update algorithms*, Inform. Process. Lett., 47 (1993), pp. 5–9.

SIAM J. COMPUT.
Vol. 27, No. 3, pp. 694–701, June 1998
© 1998 Society for Industrial and Applied Mathematics
007

COMPUTING THE ADDITIVE COMPLEXITY OF ALGEBRAIC CIRCUITS WITH ROOT EXTRACTING*

DIMA GRIGORIEV† AND MAREK KARPINSKI‡

Abstract. We design an algorithm for computing the generalized (algebraic circuits with root extracting; cf. Pippenger [*J. Comput. System Sci.*, 22 (1981), pp. 454–470], Ja'Ja' [*Proc. 22nd IEEE FOCS*, 1981, pp. 95–100], Grigoriev, Singer, and Yao [*SIAM J. Comput.*, 24 (1995), pp. 242–246]) *additive complexity* of any rational function. It is the first computability result of this sort on the additive complexity of algebraic circuits.

Key words. additive complexity, algebraic circuits, root extracting, minimal computation

AMS subject classifications. 68Q25, 68Q40, 68Q15, 26C15

PII. S0097539793258313

1. Introduction.

Whether the additive complexity of functions is computable is a well-known open problem in the theory of computation. Note that both multiplicative and total complexities of functions are computable. In this paper we prove, somewhat surprisingly, the *computability* of the generalized additive complexity for algebraic circuits with root extraction. These circuits were considered in [J 81] where a lower bound on the number of root extracting operations for computing on algebraic functions has been proven. This was recently generalized in [GSY 93] for the algebraic circuits which also contain exponential and logarithmic functions. Our result is the first computability result of this sort on the *additive* complexity of algebraic circuits.

Let us give the definition of the generalized additive complexity. $\bar{\mathbb{Q}}$ denotes the algebraic closure of $\mathbb{Q}$, the set of algebraic numbers. We say that a rational function $f \in \mathbb{Q}(X_1, \ldots, X_n)$ has a generalized additive complexity at most t if there exists a sequence of algebraic functions:

$$u_{i+1} = \varepsilon^{(i+1)} X_1^{\alpha_1^{(i+1)}} \cdots X_n^{\alpha_n^{(i+1)}} u_1^{\beta_1^{(i+1)}} \cdots u_i^{\beta_i^{(i+1)}} + \kappa^{(i+1)} X_1^{\gamma_1^{(i+1)}} \cdots X_n^{\gamma_n^{(i+1)}} u_1^{\delta_1^{(i+1)}} \cdots u_i^{\delta_i^{(i+1)}}$$

for $0 \le i \le t$, where $\kappa^{(t+1)} = 0$, $f = u_{t+1}$ and all the exponents $\alpha_1^{(i+1)}, \ldots, \delta_i^{(i+1)} \in \mathbb{Q}$, $0 \le i \le t$ are rationals, and coefficients $\varepsilon^{(i+1)}, \kappa^{(i+1)} \in \bar{\mathbb{Q}}$ are algebraic. The rationality of the exponents (rather than being integers) differs the *generalized* additive complexity from the usual *additive* complexity. In other words, we consider algebraic circuits in which (in addition to the usual arithmetic operations) extracting arbitrary roots is allowed.

If t is equal to the generalized additive complexity of f, then we say that computation $u_1, \ldots, u_{t+1}$ of f is generalized additive minimal.

In the section 2 we consider the computations in which the exponents $\alpha_1^{(i+1)}, \ldots, \delta_i^{(i+1)}$, $0 \le i \le t$ are allowed to be algebraic, and we refer to it as *the quasi-additive*

*Received by the editors October 26, 1993; accepted for publication (in revised form) April 2, 1996.

http://www.siam.org/journals/sicomp/27-3/25831.html

†Dept. of Computer Science, The Pennsylvania State University, University Park, PA 16802 (dima@cs.psu.edu). This research was partially supported by National Science Foundation grant CCR-9424358.

‡Dept. of Computer Science, University of Bonn, 53117 Bonn, and the International Computer Science Institute, Berkeley, CA (marek@cs.uni-bonn.de). This research was partially supported by DFG grant KA 673/4-1 and by the ESPRIT BR grants 7097 and EC-US030.

complexity. The computation of the quasi-additive complexity is reduced (see the lemma below) to the problem of quantifier elimination in the theory of differentially closed fields (solved in [Se 56]; for its complexity see [G 89]).

In section 3 we prove (see the proposition below) that any quasi-additive minimal computation of a rational function can be transformed into a generalized additive-minimal computation with the same number of additions which contain only rational exponents; thus quasi-additive and generalized additive complexities coincide. Moreover, the corollary in section 3 gives a possibility of constructing the rational exponents of a generalized additive-minimal computation. In section 4 we describe an algorithm for producing a generalized additive-minimal computation. In the case of one variable ($n = 1$) we give an (elementary) complexity bound of the designed algorithm (see the theorem below) as it uses the quantifier elimination algorithm from [G 89]. In the general case ($n \geq 2$) we do not give complexity bounds because the quantifier elimination method from [Se 56] is used which relies in turn on the efficient bounds in Hilbert's idealbasissatz which are not known to be elementary.

Note that first lower bounds on the additive complexity of f in terms of the variety of real roots of f were obtained in [BC 76] and [G 83] (see also [Ri 85]). One can find in [G 83] also a survey on other lower bounds, in particular on the additive complexity (see also [G 82] and [SW 80]). The lower bound from [G 83] is used (see the end of section 3) to show that there are polynomials with the generalized additive complexity equal to 3 and arbitrary large additive complexity.

Another interesting issue is the dependence of the (standard) additive complexity on the coefficients which are involved in *straight-line programs* (cf. [W 78]). If we allow only real algebraic coefficients (from $\bar{\mathbb{Q}} \cap \mathbb{R}$) instead of $\bar{\mathbb{Q}}$ (see above), then the additive complexity could jump drastically as the following example indicates. The polynomial $(1 + iX)^n + (1 - iX)^n \in \mathbb{Z}[x]$ (cf. [W 78]) *evidently* has the additive complexity at most 3 over $\bar{\mathbb{Q}}$. It also has all its roots in $\bar{\mathbb{Q}} \cap \mathbb{R}$; therefore, its *additive complexity* over $\bar{\mathbb{Q}} \cap \mathbb{R}$ is greater than $\Omega(\log^{\frac{1}{2}} n)$ (cf. [G 83], [Ri 85]).

2. Describing the quasi-additive complexity in terms of the first-order theory of differentially closed fields. We start with designing an algorithm for testing, whether there exist (and if so, also to produce) *algebraic* exponents $\alpha_1^{(i+1)}, \ldots, \delta_i^{(i+1)} \in \bar{\mathbb{Q}}$ (the algebraic closure of $\mathbb{Q}$ in $\mathbb{C}$) in the computation $u_1, \ldots, u_{t+1}$ providing an identity $u_{t+1} = f$ holds. In this case we say that f has the quasi-additive complexity at most t. For this purpose we introduce the (differential) unknowns

$$u_{i+1}, \tilde{\alpha}_1^{(i+1)}, \ldots, \tilde{\delta}_i^{(i+1)}, v_1^{(i+1)}, \ldots, v_n^{(i+1)}, w_1^{(i+1)}, \ldots, w_i^{(i+1)},$$
$$\tilde{v}_1^{(i+1)}, \ldots, \tilde{v}_n^{(i+1)}, \tilde{w}_1^{(i+1)}, \ldots, \tilde{w}_i^{(i+1)}$$

for all $0 \leq i \leq t$ and the system of (partial) differential equations (denote $D_i = \frac{d}{dX_i}$ and by D any of the operators $D_1, \ldots, D_n$; denote the Kronecker symbol by $\delta(l, j)$):

$(1a)_{i+1}$ $\quad D(\tilde{\alpha}_1^{(i+1)}) = \cdots = D(\tilde{\delta}_i^{(i+1)}) = 0,$

$(1b)_{i+1}$ $\quad D_j(v_l^{(i+1)}) = \frac{\tilde{\alpha}_l^{(i+1)}}{X_l} v_l^{(i+1)} \delta(l, j), \qquad D_j(\tilde{v}_l^{(i+1)}) = \frac{\tilde{\gamma}_l^{(i+1)}}{X_l} \tilde{v}_l^{(i+1)} \delta(l, j),$
$1 \leq l, j \leq n,$

$(1c)_{i+1}$ $\quad D(w_l^{(i+1)}) = \tilde{\beta}_l^{(i+1)} w_l^{(i+1)} \frac{Du_l}{u_l}, \qquad D(\tilde{w}_l^{(i+1)}) = \tilde{\delta}_l^{(i+1)} \tilde{w}_l^{(i+1)} \frac{Du_l}{u_l},$
$1 \leq l \leq i,$

$(1d)_{i+1}$ $\quad u_{i+1} = v_1^{(i+1)} \cdots v_n^{(i+1)} w_1^{(i+1)} \cdots w_i^{(i+1)} + \tilde{v}_1^{(i+1)} \ldots \tilde{v}_n^{(i+1)} \tilde{w}_1^{(i+1)} \ldots \tilde{w}_i^{(i+1)}$

for all $0 \le i \le t$ together with the equation $u_{t+1} = f$. The resulting system we denote by (1).

Note that the equations $(1a)_{i+1}$ imply that $\tilde{\alpha}_1^{(i+1)}, \ldots, \tilde{\delta}_i^{(i+1)} \in \bar{\mathbb{Q}}$ are the constants; $(1b)_{i+1}$ imply that $v_l^{(i+1)} = \mu_l^{(i+1)} X_l^{\tilde{\alpha}_l^{(i+1)}}, \tilde{v}_l^{(i+1)} = \tilde{\mu}_l^{(i+1)} X_l^{\tilde{\gamma}_l^{(i+1)}}$ for the appropriate constants $\mu_l^{(i+1)}, \tilde{\mu}_l^{(i+1)} \in \bar{\mathbb{Q}}$; $(1c)_{(i+1)}$ imply that $w_l^{(i+1)} = \nu_l^{(i+1)} u_l^{\tilde{\beta}_l^{(i+1)}}, \tilde{w}_l^{(i+1)} = \tilde{\nu}_l^{(i+1)} u_l^{\tilde{\delta}_l^{(i+1)}}$ for the appropriate constants $\nu_l^{(i+1)}, \tilde{\nu}_l^{(i+1)} \in \bar{\mathbb{Q}}$.

Thus, the following lemma is proved.

LEMMA. *The solvability of system* (1) (*in all its differential unknowns*) *is equivalent to the fact that the quasi-additive complexity of* f *is at most* t.

Now we consider the statement of solvability of the system (1) as an existential formula of the first-order theory of differentially closed fields [Se 56]. Applying to it a quantifier elimination algorithm [Se 56] one can eliminate unknowns

$$u_{i+1}, v_1^{(i+1)}, \ldots, v_n^{(i+1)}, w_1^{(i+1)}, \ldots, w_i^{(i+1)}, \tilde{v}_1^{(i+1)}, \ldots, \tilde{v}_n^{(i+1)}, \tilde{w}_1^{(i+1)}, \ldots, \tilde{w}_i^{(i+1)} \quad \text{for all } 0 \le i \le t.$$

As a result we get an (existential) equivalent formula containing only the unknowns $\tilde{\alpha}_1^{(i+1)}, \ldots, \tilde{\delta}_i^{(i+1)}, 0 \le i \le t$. Because of (1a) the latter formula can be considered as a formula in the language of polynomials (so, without derivatives), thus as a system of polynomial equations and inequalities with integer coefficients.

Thus, given a rational function f the algorithm tries $t = 1, 2, \ldots$, and for each t tests (using [CG 83], [C 86]) whether the above constructed system of polynomial equations and inequalities has a solution (over $\bar{\mathbb{Q}}$). For a minimal such t we take any of these solutions $\alpha_1^{(i+1)}, \ldots, \delta_i^{(i+1)} \in \bar{\mathbb{Q}}, 0 \le i \le t$. In the next section we show that in this case there exists as well a rational solution of this system, and moreover we show how to construct it.

To solve the system (1) of differential equations we applied the algorithm from [Se 56] for which the elementary complexity bound is unknown since it relies on an efficient bound in Hilbert's idealbasissatz. But the complexity of quantifier elimination is elementary in the case of ordinary differential equations for the algorithm designed in [G 89], i.e., when $n = 1$, and in other words when there is only one independent variable X. In this case system (1) contains $O(t^2)$ unknowns, the order of highest occurring derivatives in the equations is at most 1, the degree of the equations is at most $O(t) + \deg f$ and the number of equations is at most $O(t^2)$, and the bit-size of the coefficients of the occurring equations is at most $O(1) + M$, where M is the bit-size of the coefficients of f. Therefore (see the bounds in [G 89]), one can eliminate quantifiers and produce a system of polynomial equations and inequalities with integer coefficients (see above) in the unknowns $\tilde{\alpha}_1^{(i+1)}, \ldots, \tilde{\delta}_i^{(i+1)}, 0 \le i \le t$ in time $\mathcal{N} = M^{O(1)} (\deg f)^{2^{2^{O(t^2)}}}$; the degrees of the polynomials occurring in this system do not exceed $\mathcal{N}_1 = (\deg f)^{2^{2^{O(t^2)}}}$, the number of these polynomials is at most $\mathcal{N}_1$, and the bit-size of (integer) coefficients occurring in this system can be bounded by $\mathcal{N}$.

Therefore, to solve this system of polynomial equations and inequalities we apply the algorithm from [CG 83] (cf. also [C 86]) which requires time $M^{O(1)} (\deg f)^{2^{2^{O(t^2)}}}$. The algorithm from [CG 83] finds (provided that the system is solvable) a solution $\alpha_1^{(i+1)}, \ldots, \delta_i^{(i+1)} \in \bar{\mathbb{Q}}, 0 \le i \le t$ in the following form. The algorithm produces an irreducible over $\mathbb{Q}$ polynomial $\varphi(Z) \in \mathbb{Q}[Z]$ and polynomials $\bar{\alpha}_1^{(i+1)}(Z), \ldots, \bar{\delta}_i^{(i+1)}(Z) \in$

$\mathbb{Q}[Z], 0 \le i \le t$ such that $\alpha_1^{(i+1)} = \bar{\alpha_1}^{(i+1)}(\theta), \dots, \delta_i^{(i+1)} = \bar{\delta}_i^{(i+1)}(\theta)$, where $\theta \in \bar{\mathbb{Q}}$ is a root of $\varphi(\theta) = 0$. From [CG 83] we obtain the following bounds:

$$\deg(\varphi), \deg(\bar{\alpha}_1^{(i+1)}), \dots, \deg(\bar{\delta}_i^{(i+1)}) \le (\deg f)^{2^{2^{O(t^2)}}}, 0 \le i \le t,$$

and the bit-size of every coefficient occurring in the listed polynomials does not exceed $M^{O(1)}\ (\deg f)^{2^{2^{O(t^2)}}}$.

3. Rational exponents in the quasi-additive minimal computation. In this section we prove (see the proposition below) the equivalence of the generalized additive and quasi-additive complexities for rational functions. Moreover, we show (see the corollary below) for given algebraic exponents of a quasi-additive minimal computation how to produce the exponents of a certain generalized additive-minimal computation of the same rational function, thus containing only rational exponents. The similar statements were proved also for the rationality of the exponents in the minimal sparse representations of a rational function [GKS 92a] and of a real algebraic function [GKS 92b]. But the latter statements have a nature different from the one in the present paper; another difference is that we prove here the existence of the rational exponents rather than the rationality as it was the case in [GKS 92a], [GKS 92b].

So, let

$$u_{i+1} = \varepsilon^{(i+1)} X_1^{\alpha_1^{(i+1)}} \cdots X_n^{\alpha_n^{(i+1)}} u_1^{\beta_1^{(i+1)}} \cdots u_i^{\beta_i^{(i+1)}} + \kappa^{(i+1)} X_1^{\gamma_1^{(i+1)}} \cdots X_n^{\gamma_n^{(i+1)}} u_1^{\delta_1^{(i+1)}} \cdots u_i^{\delta_i^{(i+1)}},$$

where $0 \le i \le t, \kappa^{(t+1)} = 0$ and all the exponents and coefficients

$$\alpha_1^{(i+1)}, \dots, \delta_i^{(i+1)}, \varepsilon^{(i+1)}, \kappa^{(i+1)} \in \bar{\mathbb{Q}}\,.$$

PROPOSITION. *Assume that $f = u^{(t+1)} \in \bar{\mathbb{Q}}(X_1, \dots, X_n)$ is a rational function and t is the minimal possible (so t is equal to the quasi-additive complexity of f). Then there exist rational exponents $a_1^{(i+1)}, \dots, d_i^{(i+1)} \in \mathbb{Q}, 0 \le i \le t$, respectively, providing also a computation of f (thus, t is also equal to the generalized additive complexity).*

Proof. For each $1 \le j \le n$ consider a $\mathbb{Q}$-basis $\bar{\theta}_j^{(1)}, \bar{\theta}_j^{(2)}, \dots \in \bar{\mathbb{Q}}$ of the $\mathbb{Q}$-linear hull $\mathbb{Q}\{\alpha_j^{(s)}, \gamma_j^{(s)}\}_{1 \le s \le t+1}$. If 1 (thereby $\mathbb{Q}$) is contained in the latter linear hull, then we set $\bar{\theta}_j^{(1)} = 1$. Denote $\{\theta_j^{(1)}, \theta_j^{(2)}, \dots\} = \{\bar{\theta}_j^{(1)}, \bar{\theta}_j^{(2)}, \dots\} \setminus \{1\}$.

Consider a differential field $F_j, 0 \le j \le n$ generated over $\bar{\mathbb{Q}}(X_1, \dots, X_n)$ by the elements $\log X_1, X_1^{\theta_1^{(1)}}, X_1^{\theta_1^{(2)}}, \dots, \log X_j, X_j^{\theta_j^{(1)}}, X_j^{\theta_j^{(2)}}, \dots$. Then in the terminology of [RC 79] each $F_j, 0 \le j \le n$ is a log-explicit extension of its field of constants $\bar{\mathbb{Q}}$ (one can represent $X^\beta = \exp(\beta \log X)$).

We claim that the elements $X_{j+1}^{\theta_{j+1}^{(1)}}, X_{j+1}^{\theta_{j+1}^{(2)}}, \dots \in F_{j+1}$ are algebraically independent over the field $F_j(\log X_{j+1})$. Assume the contrary. Then Corollary 3.2 of [RC 79] (see also [Ro 76]) implies the existence of a constant $\kappa \in \bar{\mathbb{Q}}$, rational numbers

$$l_1^{(0)}, l_1^{(1)}, \dots, l_j^{(0)}, l_j^{(1)}, \dots, l_{j+1}^{(0)}, l_{j+1}^{(1)}, \dots \in \mathbb{Q}$$

such that not all $l_{j+1}^{(1)}, l_{j+1}^{(2)}, \dots$ are zeros, and

$$X_{j+1}^{l_{j+1}^{(0)} + \Sigma_{k \ge 1} l_{j+1}^{(k)} \theta_{j+1}^{(k)}} = \kappa X_1^{l_1^{(0)} + \Sigma_{k \ge 1} l_1^{(k)} \theta_1^{(k)}} \cdots X_j^{l_j^{(0)} + \Sigma_{k \ge 1} l_j^{(k)} \theta_j^{(k)}},$$

but this leads to a contradiction since the derivative $\frac{d}{dX_{j+1}}$ of the left side is nonzero, but the derivative of the right side is equal to zero.

For each $1 \le i \le t$ consider a $\mathbb{Q}$-basis $\bar\eta_i^{(1)}, \bar\eta_i^{(2)}, \ldots \in \bar{\mathbb{Q}}$ of the $\mathbb{Q}$-linear hull $\mathbb{Q}\{\beta_i^{(s)}, \delta_i^{(s)}\}_{i+1\le s\le t+1}$. If 1 (thereby $\mathbb{Q}$) is contained in the latter linear hull, then we set $\bar\eta_i^{(1)} = 1$. Denote $\{\eta_i^{(1)}, \eta_i^{(2)}, \ldots\} = \{\bar\eta_i^{(1)}, \bar\eta_i^{(2)}, \ldots\} \setminus \{1\}$.

Denote by E_i, $0 \le i \le t$ a field generated over F_n by the elements

$$\log u_1, u_1^{\eta_1^{(1)}}, u_1^{\eta_1^{(2)}}, \ldots, \log u_i, u_i^{\eta_i^{(1)}}, u_i^{\eta_i^{(2)}}, \ldots .$$

It is a log-explicit extension of its field of constants $\bar{\mathbb{Q}}$.

We claim that for $0 \le i \le t-1$ the elements $u_{i+1}^{\eta_{i+1}^{(1)}}, u_{i+1}^{\eta_{i+1}^{(2)}}, \ldots \in E_{i+1}$ are algebraically independent over the field $E_i(\log u_{i+1})$. Assume the contrary. Then again using Corollary 3.2 of [RC 79] we conclude that there exists a constant $\varepsilon \in \bar{\mathbb{Q}}$, rational numbers

$$p_1, p_1^{(1)}, p_1^{(2)}, \ldots, p_n, p_n^{(1)}, p_n^{(2)}, \ldots, z_1, z_1^{(1)}, z_1^{(2)}, \ldots, z_{i+1}, z_{i+1}^{(1)}, z_{i+1}^{(2)}, \ldots \in \mathbb{Q}$$

such that not all $z_{i+1}^{(1)}, z_{i+1}^{(2)}, \ldots$ are zeros and

$$u_{i+1}^{z_{i+1}+\Sigma_{j\ge1} z_{i+1}^{(j)}\eta_{i+1}^{(j)}} = \varepsilon X_1^{p_1+\Sigma_{j\ge1}p_1^{(j)}\theta_1^{(j)}} \cdots X_n^{p_n+\Sigma_{j\ge1}p_n^{(j)}\theta_n^{(j)}} u_1^{z_1+\Sigma_{j\ge1}z_1^{(j)}\eta_1^{(j)}} \cdots u_i^{z_i+\Sigma_{j\ge1}z_i^{(j)}\eta_i^{(j)}}.$$

This provides an expression of u_{i+1} as a product of powers of $X_1, \ldots, X_n, u_1, \ldots, u_i$, and thereby we can diminish t by 1 in the computation of f; this contradiction with the minimality of t proves the algebraic independence of $u_{i+1}^{\eta_{i+1}^{(1)}}, u_{i+1}^{\eta_{i+2}^{(2)}}, \ldots$ over $E_i(\log u_{i+1})$.

Consider the expansions

$$\begin{aligned}
\alpha_j^{(s)} &= a_j^{(s)} + \sum_{k\ge1} a_{j,k}^{(s)}\theta_j^{(k)}, \gamma_j^{(s)} = \quad c_j^{(s)} + \sum_{k\ge1} c_{j,k}^{(s)}\theta_j^{(k)}, \quad 1 \le j \le n, \qquad 1 \le s \le t+1,\\
\beta_i^{(s)} &= b_i^{(s)} + \sum_{k\ge1} b_{i,k}^{(s)}\eta_i^{(k)}, \delta_i^{(s)} = \quad d_i^{(s)} + \sum_{k\ge1} d_{i,k}^{(s)}\theta_i^{(k)}, \quad 1 \le i \le t+1, \quad i < s \le t+1,
\end{aligned} \tag{2}$$

where $a_j^{(s)}, \ldots, d_{i,k}^{(s)} \in \mathbb{Q}$ are suitable rationals. Remark that if $1 \notin \{\bar\theta_j^{(1)}, \bar\theta_j^{(2)}, \ldots\}$, then $a_j^{(s)} = c_j^{(s)} = 0$; also if $1 \notin \{\bar\eta_i^{(1)}, \bar\eta_i^{(2)}, \ldots\}$, then $b_i^{(s)} = d_i^{(s)} = 0$. Then we can rewrite the initial computation $u_1, u_2, \ldots$ as follows:

$$\begin{aligned}
u_{i+1} = \varepsilon^{(i+1)} & X_1^{a_1^{(i+1)}} (X_1^{\theta_1^{(1)}})^{a_{1,1}^{(i+1)}} (X_1^{\theta_1^{(2)}})^{a_{1,2}^{(i+1)}} \cdots X_n^{a_n^{(i+1)}} (X_n^{\theta_n^{(1)}})^{a_{n,1}^{(i+1)}} \cdots \\
& u_1^{b_1^{(i+1)}} (u_1^{\eta_1^{(1)}})^{b_{1,1}^{(i+1)}} (u_1^{\eta_1^{(2)}})^{b_{1,2}^{(i+1)}} \cdots u_i^{b_i^{(i+1)}} (u_i^{\eta_i^{(1)}})^{b_{i,1}^{(i+1)}} (u_i^{\eta_i^{(2)}})^{b_{i,2}^{(i+1)}} \cdots \\
& + \kappa^{(i+1)} X_1^{c_1^{(i+1)}} (X_1^{\theta_1^{(1)}})^{c_{1,1}^{(i+1)}} (X_1^{\theta_1^{(2)}})^{c_{1,2}^{(i+1)}} \cdots X_n^{c_n^{(i+1)}} (X_n^{\theta_n^{(1)}})^{c_{n,1}^{(i+1)}} \cdots \\
& u_1^{d_1^{(i+1)}} (u_1^{\eta_1^{(1)}})^{d_{1,1}^{(i+1)}} (u_1^{\eta_1^{(2)}})^{d_{1,2}^{(i+1)}} \cdots u_i^{d_i^{(i+1)}} (u_i^{\eta_i^{(1)}})^{d_{i,1}^{(i+1)}} (u_i^{\eta_i^{(2)}})^{d_{i,2}^{(i+1)}} \cdots .
\end{aligned} \tag{3}$$

From the latter expression one can show by induction on i that u_{i+1} (and thereby each of the previous elements $u_1, \ldots, u_i$) is algebraic over the field $E_i' \subset E_i$ generated over $\bar{\mathbb{Q}}(X_1, \ldots, X_n)$ by the elements

$$X_1^{\theta_1^{(1)}}, X_1^{\theta_1^{(2)}}, \ldots, X_n^{\theta_n^{(1)}}, X_n^{\theta_n^{(2)}}, \ldots, u_1^{\eta_1^{(1)}}, u_1^{\eta_1^{(2)}}, \ldots, u_i^{\eta_i^{(1)}}, u_i^{\eta_i^{(2)}}, \ldots .$$

Above we have proved that the latter elements are algebraically independent over $\bar{\mathbb{Q}}(X_1,\dots,X_n)$. Because $u_{t+1}=f\in\bar{\mathbb{Q}}(X_1,\dots,X_n)$ we can substitute in the expression (3) instead of the elements

$$X_1^{\theta_1^{(1)}},X_1^{\theta_1^{(2)}},\dots,X_n^{\theta_n^{(1)}},X_n^{\theta_n^{(2)}},\dots,u_1^{\eta_1^{(1)}},u_1^{\eta_1^{(2)}},\dots,u_t^{\eta_t^{(1)}},u_t^{\eta_t^{(2)}},\dots$$

almost (in the sense of Zariski topology) arbitrary constants

$$y_1^{(1)},y_1^{(2)},\dots,y_n^{(1)},y_n^{(2)},\dots,z_1^{(1)},z_1^{(2)},\dots,z_t^{(1)},z_t^{(2)},\dots\in\bar{\mathbb{Q}},$$

respectively, with the mere requirement that in the intermediate computations of $u_1,u_2,\dots,u_{t+1}=f$ no nonpositive powers of zero are taken (each time we choose some branch of a rational power).

As a result we get a computation of $\tilde{u}_1,\tilde{u}_2,\dots,\tilde{u}_{t+1}=f$ in which only rational exponents occur; namely,

$$\tilde{u}_{i+1}=\tilde{\varepsilon}^{(i+1)}X_1^{a_1^{(i+1)}}\cdots X_n^{a_n^{(i+1)}}\tilde{u}_1^{b_1^{(i+1)}}\cdots\tilde{u}_i^{b_i^{(i+1)}}+\tilde{\kappa}^{(i+1)}X_1^{c_1^{(i+1)}}\cdots X_n^{c_n^{(i+1)}}\tilde{u}_1^{d_1^{(i+1)}}\cdots\tilde{u}_i^{d_i^{(i+1)}} \tag{4}$$

for some $\tilde{\varepsilon}^{(i+1)},\ \tilde{\kappa}^{(i+1)}\in\bar{\mathbb{Q}}$. The proposition is proved.

From the proof of the proposition we extract the following corollary.

COROLLARY. *For every* $1\le i\le t,\ 1\in\mathbb{Q}\{\beta_i^{(s)},\delta_i^{(s)}\}_{i+1\le s\le t+1}$. *For any* $\mathbb{Q}$*-basis* $\bar{\theta}_j^{(1)},\bar{\theta}_j^{(2)},\dots$ *of* $\mathbb{Q}\{\alpha_j^{(s)},\gamma_j^{(s)}\}_{1\le s\le t+1}$ *and any* $\mathbb{Q}$*-basis* $\bar{\eta}_i^{(1)},\bar{\eta}_i^{(2)},\dots$ *of* $\mathbb{Q}\{\beta_i^{(s)},\delta_i^{(s)}\}_{i+1\le s\le t+1}$ *we get the rational exponents of the resulting computation of* $\tilde{u}_1,\dots,\tilde{u}_{t+1}$ *(see* (4)*) from the expansions* (2).

In order to show that $1\in\mathbb{Q}\{\beta_i^{(s)},\delta_i^{(s)}\}_s$, observe that otherwise $b_i^{(s)}=d_i^{(s)}=0$ for all $i+1\le s\le t+1$, and we could diminish t by deleting $\tilde{u}_i$ from the computation $\tilde{u}_1,\dots,\tilde{u}_{t+1}$ and get a contradiction with a minimality of t.

Remark that the corollary together with Lemma 12 of [GKS 92a] entail that for any i the constructible set of all the possible exponent vectors $(\beta_i^{(i+1)},\dots,\beta_i^{(t+1)},\delta_i^{(i+1)},\dots,\delta_i^{(t)})\in\bar{\mathbb{Q}}^{2t-2i+1}$ is contained in a finite union of the hyperplanes of the kind

$$\sum_{i+1\le j\le t+1}\hat{b}_i^{(j)}\beta_i^{(j)}+\sum_{i+1\le j\le t}\hat{d}_i^{(j)}\delta_i^{(j)}=\hat{d},$$

where $\hat{b}_i^{(j)},\hat{d}_i^{(j)},\hat{d}\in\mathbb{Z}$. The similar case also holds for the vectors $(\alpha_i^{(1)},\dots,\alpha_i^{(t+1)},\gamma_i^{(1)},\dots,\gamma_i^{(t)})\in\bar{\mathbb{Q}}^{2t+1}$. But we will not use this remark.

Note also that in the resulting computation (4) the rational exponents depend on the choice of the $\mathbb{Q}$-basis (see the corollary). The following simple example demonstrates that the dependency really can happen:

$$u_1=X^\alpha(X+1),u_2=X^{-a\alpha}u_1^a+X^{-b\alpha}u_1^b=(X+1)^a+(X+1)^b,$$

where $\alpha\in\bar{\mathbb{Q}}\setminus\mathbb{Q},a,b\in\mathbb{Q}$. Choosing a basis $\alpha+z,1\in\mathbb{Q}\{1,\alpha\}$ for arbitrary $z\in\mathbb{Q}$, we get

$$\begin{aligned}u_1&=(X^{\alpha+z})X^{1-z}+(X^{\alpha+z})X^{-z},\\ u_2&=(X^{\alpha+z})^{-a}X^{za}u_1^a+(X^{\alpha+z})^{-b}X^{zb}u_1^b,\end{aligned}$$

and by the corollary we get

$$\begin{aligned}u_1&=wX^{1-z}+wX^{-z},\\ u_2&=w^{-a}X^{za}u_1^a+w^{-b}X^{zb}u_1^b\end{aligned}$$

for arbitrary $w\in\bar{\mathbb{Q}}\setminus\{0\}$.

4. Constructing a generalized additive-minimal computation. The previous two sections (see the lemma and corollary) give us a possibility of computing a generalized additive complexity t of a rational function f. Now we complete an algorithm which finds some generalized additive-minimal circuit computing f. Using the corollary from section 3, the algorithm finds rational exponents $\alpha_1^{(i+1)}, \ldots, \delta_i^{(i+1)} \in \mathbb{Q}, 0 \leq i \leq t$; it remains to find the coefficients $\varepsilon^{(i+1)}, \kappa^{(i+1)} \in \bar{\mathbb{Q}}, 0 \leq i \leq t$.

Denote by $\mathcal{M}$ a bound on the bit-sizes of the rational exponents $\alpha_1^{(i+1)}, \ldots, \delta_i^{(i+1)} \in \mathbb{Q}, 0 \leq i \leq t$. Then by induction on i one can easily show that each $u_{i+1}, v_1^{(i+1)}, \ldots, \tilde{w}_i^{(i+1)},\ 0 \leq i \leq t$ is an algebraic function of the degree (i.e., the degree of a minimal polynomial which satisfies the function) at most $N = (\exp(\mathcal{M}))^{t^{O(t)}}$. Hence the coefficients $\varepsilon^{(i+1)}, \kappa^{(i+1)},\ 0 \leq i \leq t$ fit if and only if for every $1 \leq x_1, \ldots, x_n \leq N^2$ for which all the intermediate computations of the circuit are definable, the equality $u_{t+1}(x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$ holds. So, for every fixed $1 \leq x_1, \ldots, x_n \leq N^2$ we introduce the variables

$$u_{t+1}(x_1, \ldots, x_n), v_1^{(i+1)}(x_1, \ldots, x_n), \ldots, \tilde{w}_i^{(i+1)}(x_1, \ldots, x_n),\ 0 \leq i \leq t$$

and write down a system of polynomial equations and inequalities expressing all the operations of the circuit (provided that they are all definable) and finally the relation $u_{t+1}(x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$. Then the algorithm invoking [CG 83] solves this system in $N^{2n} + 2t + 1$ variables and finds in particular $\varepsilon^{(i+1)}, \kappa^{(i+1)} \in \bar{\mathbb{Q}}, 0 \leq i \leq t$. More precisely, for each subset $J \subset \{1, \ldots, N^2\}^n$ we consider a system as above including in it just the points $(x_1, \ldots, x_n) \in J$ (so J plays the role of the set of points in which the computation is defined). The algorithm solves this system and takes J with the maximal cardinality for which the system is solvable. In a more sophisticated way we can partition the cube $\{1, \ldots, N^2\}^n$ into N^n subcubes with sides equal to N and as J take each of these subcubes, but this improvement does not change the complexity bounds below.

In the ordinary case ($n = 1$) we can bound the complexity of the described algorithm. First, observe that in this case $\mathcal{M} \leq M^{O(1)}(\deg f)^{2^{2^{O(t^2)}}}$ (see the end of section 1). Therefore, the system of polynomial equations and inequalities constructed above contains $\exp(M^{O(1)}(\deg f)^{2^{2^{O(t^2)}}})$ polynomials of degrees at most $\exp(M^{O(1)}(\deg f)^{2^{2^{O(t^2)}}})$ in $\exp(M^{O(1)}(\deg f)^{2^{2^{O(t^2)}}})$ variables. Hence one can solve it using the algorithm from [CG 83] in time $\exp(\exp(M^{O(1)}(\deg f)^{2^{2^{O(t^2)}}}))$ and find $\varepsilon^{(i+1)}, \kappa^{(i+1)} \in \bar{\mathbb{Q}}$, $0 \leq i \leq t$, representing them as algebraic numbers as at the end of section 1 with the size also bounded by the latter value.

Summarizing, we formulate the following theorem.

THEOREM. (a) *There is an algorithm calculating the generalized additive complexity of a rational function $f \in \mathbb{Q}(x_1, \ldots, x_n)$ and constructing a generalized additive-minimal circuit computing f.*

(b) *In the case of one-variable rational functions f the running time of the algorithm from* (a) *can be bounded by* $\exp(\exp(M^{O(1)}(\deg f)^{2^{2^{O(t^2)}}}))$, *where M bounds the bit-size of each* (*rational*) *coefficient of f. The absolute values of the numerators and denominators of the found rational exponents in a generalized additive-minimal circuit computing f do not exceed* $\exp(M^{O(1)}(\deg f)^{2^{2^{O(t^2)}}})$.

At the end we demonstrate that there could be a big gap between the additive complexity and generalized additive complexity. Consider a polynomial

$$f_n = (1 + X^{\frac{1}{2}})^n + (1 - X^{\frac{1}{2}})^n \in \mathbb{Z}[X]$$

with the generalized additive complexity at most 3. Because all its $\lfloor \frac{n}{2} \rfloor$ roots are negative reals, the additive complexity of f_n is at least $\Omega((\log n)^{\frac{1}{2}})$ because of the result of [G 83] (see also [Ri 85]) based on the method from [Kh 91].

5. Further research. It remains an interesting open problem on improving the complexity bounds of our algorithm. Also it will be very interesting to shed some more light on the status of the problem of computing the *standard additive* complexity of rational functions. At this point we do not know much about this problem.

Acknowledgments. We are thankful to Richard Cleve for starting us thinking about the additive complexity of polynomials and to Allan Borodin, Joachim von zur Gathen, Thomas Lickteig, Michael Singer, Volker Strassen, and Andy Yao for many interesting discussions.

REFERENCES

[BC 76] A. Borodin and S. Cook, *On the number of additions to compute specific polynomials*, SIAM J. Comput., 5 (1976), pp. 146–157.

[C 86] A. Chistov, *Algorithms of polynomial complexity for factoring polynomials and finding the components of varieties in subexponential time*, J. Soviet Math., 34 (1986), pp. 1838–1882.

[CG 83] A. Chistov and D. Grigoriev, *Subexponential-time solving systems of algebraic equations*, LOMI Preprints E-9-83, E-10-83, Steklov Math. Institute, 1983.

[G 82] D. Grigoriev, *Additive complexity in directed computations*, Theoret. Comput. Sci., 19 (1982), pp. 39–67.

[G 83] D. Grigoriev, *Lower bounds in algebraic complexity*, Transl. in J. Soviet Math., 29 (1985), pp. 1388–1425.

[G 89] D. Grigoriev, *Complexity of quantifier elimination in the theory of differential equations*, Lecture Notes in Comput. Sci., 378 (1989), pp. 11–25.

[GK 91] D. Grigoriev and M. Karpinski, *Algorithms for sparse rational interpolation*, in Proc. ACM ISSAC, 1991, pp. 7–13.

[GKS 90] D. Grigoriev, M. Karpinski, and M. Singer, *Interpolation of sparse rational functions without knowing bounds on exponents*, in Proc. 31st IEEE FOCS, 1990, pp. 840–846.

[GKS 92a] D. Grigoriev, M. Karpinski, and M. Singer, *Computational complexity of sparse rational interpolation*, SIAM J. Comput., 23 (1994), pp. 1–11.

[GKS 92b] D. Grigoriev, M. Karpinski, and M. Singer, *Computational complexity of sparse real algebraic function interpolation*, Progr. Math., 109 (1993), pp. 91–104.

[GSY 93] D. Grigoriev, M. Singer, and A. Yao, *On computing algebraic functions using logarithms and exponentials*, SIAM J. Comput., 24 (1995), pp. 242–246.

[J 81] J. Ja'Ja', *Computations of algebraic functions with root extractions*, in Proc. 22nd IEEE FOCS, 1981, pp. 95–100.

[KW 93] M. Karpinski and T. Werther, *VC dimension and uniform learnability of sparse polynomials and rational functions*, SIAM J. Comput., 22 (1993), pp. 1276–1285.

[Kh 91] A. Khovanski, *Fewnomials*, AMS Transl. Math. Monogr. 88, 1991.

[P 81] N. Pippenger, *Computational complexity of algebraic functions*, J. Comput. System Sci., 22 (1981), pp. 454–470.

[Ri 85] J. J. Risler, *Additive complexity and zeros of real polynomials*, SIAM J. Comput., 14 (1985), pp. 178–183.

[Ro 76] M. Rosenlicht, *On Liouville's theory of elementary functions*, Pacific J. Math., 65 (1976), pp. 485–492.

[RC 79] M. Rothstein and B. Caviness, *A structure theorem for exponential and primitive functions*, SIAM J. Comput., 8 (1979), pp. 357–366.

[Se 56] A. Seidenberg, *An elimination theory for differential algebra*, Univ. of Calif. Press, 3, N 2, (1956), pp. 31–66.

[SW 80] C. Schnorr and J. van de Wiele, *On the additive complexity of polynomials*, Theoret. Comput. Sci., 10 (1980), pp. 1–18.

[W 78] J. P. van de Wiele, *Complexité additive et zéros des polynômes à coefficients réels et complexes*, Rapport de Recherche Laboria N$^{\text{O}}$ 292, Mars 1978.

SIAM J. COMPUT.
Vol. 27, No. 3, pp. 702–712, June 1998
© 1998 Society for Industrial and Applied Mathematics
008

AN $\Omega(D\log(N/D))$ LOWER BOUND FOR BROADCAST IN RADIO NETWORKS*

EYAL KUSHILEVITZ† AND YISHAY MANSOUR‡

Abstract. We show that for any *randomized* broadcast protocol for radio networks, there exists a network in which the expected time to broadcast a message is $\Omega(D\log(N/D))$, where D is the diameter of the network and N is the number of nodes. This implies a tight lower bound of $\Omega(D\log N)$ for any $D \leq N^{1-\varepsilon}$, where $\varepsilon > 0$ is any constant.

Key words. radio networks, broadcast, lower bounds

AMS subject classifications. 68Q22, 68M10

PII. S0097539794279109

1. Introduction.

Traditionally, radio networks have received considerable attention due to their military significance. The growing interest in cellular telephones and wireless communication networks has reinforced the interest in radio networks. The basic feature of radio networks, which distinguishes them from other networks, is that a processor can receive a message only from a *single* neighbor at a certain time. If two (or more) neighbors of a processor transmit concurrently, then the processor would not receive either messages.

In many applications, the users of the radio network are mobile, and therefore the topology is unstable. For this reason, it is desirable for radio-networks algorithms to refrain from making assumptions about the network topology, or about the information that processors have concerning the topology. In this work we assume that none of the processors initially have any topological information, except for the size of the network and its diameter.[1] See [Tan81, Gal85, BGI92, BGI91] for a discussion on this model and related models.

We study *broadcast* protocols; those protocols are initiated by a single processor (the *originator*) that has a message M it wishes to propagate to all the other processors in the network. In many of the radio-networks applications (e.g., cellular phones) broadcast is a central primitive which is frequently used, for example, to perform a network-wide search for a user.

Bar-Yehuda, Goldreich, and Itai [BGI92] present a *randomized* broadcast algorithm, that runs in expected $O(D\log N + \log^2 N)$ time slots, where N is the number of processors in the network and D is its diameter. In contrast, they show that for any *deterministic* broadcast algorithm there are networks of constant diameter on which the algorithm needs $\Omega(N)$ time slots.

*Received by the editors September 8, 1994; accepted for publication (in revised form) April 3, 1996. An early version of this paper appeared in Proc. 12th ACM Symp. on Principles of Distributed Computing, ACM, New York, 1993, pp. 65–74.

http://www.siam.org/journals/sicomp/27-3/27910.html

†Aiken Computation Lab., Harvard University, Cambridge, MA 02138-2901. Current address: Dept. of Computer Science, Technion, Haifa, 32000, Israel (eyalk@cs.technion.ac.il). This research was supported by research contracts ONR-N0001491-J-1981 and NSF-CCR-90-07677.

‡Computer Science Dept., Tel-Aviv University, Ramat Aviv, Tel Aviv 69978, Israel and IBM – T. J. Watson Research Center (mansour@cs.tau.ac.il).

[1]Usually, when the topology is unstable, the diameter is unknown to the processors and only a bound on the size of the network is available. However, since we are proving a *lower bound*, this assumption only makes the result stronger.

Alon et al. [ABLP91] made the first step towards proving the optimality of the upper bound of [BGI92]. Their result can be viewed as a graph-theoretic result; they show that there exist networks of diameter $D = 3$ on which any schedule needs at least $\Omega(\log^2 N)$ time slots. This lower bound shows that there are networks on which broadcast requires this many time slots, and it matches the known upper bounds [BGI92, CW87], in the case of constant-diameter networks.

In this work we complete the picture by proving an $\Omega(D \log(N/D))$ lower bound. Our result is of a different nature; we show that for any *randomized* broadcast algorithm and parameters N and D, there is an ordering of the N processors in a network of diameter D such that the expected number of time slots, used by the algorithm, is $\Omega(D \log(N/D))$. For $D \leq N^{1-\varepsilon}$ this gives an $\Omega(D \log N)$ lower bound. Hence, it proves the tightness of the upper bound of [BGI92] for all N and $D \leq N^{1-\varepsilon}$. Moreover, the lower bound holds even if each of the N processors is allowed to use a *different* program (e.g., the processors can use their IDs). In a recent work, Gaber and Mansour [GM95] have shown that for every network, there exists a schedule whose time is $O(D + \log^5 N)$. The scheduler there needs to get the topology of the network in advance, in order to build the schedule. The result of [GM95] shows that the lower bound presented here must rely heavily on the lack of topological knowledge at the processors.

Broadcast in radio networks has received considerable attention in previous works. [CW87] present a deterministic sequential algorithm that, given the network, finds in polynomial time a legal schedule that requires at most $O(D \log^2 N)$ time slots. Broadcast that is based on using a spanning tree was suggested in [CK85a, CK87]. In [BII93] it is shown how to reduce the amortized cost per broadcast by using a breadth-first-search (BFS) tree. Simulation of point to point networks on radio networks is found in [CK85b, ABLP92, BGI91].

An important issue in the study of radio networks is whether collisions can be detected; namely, whether a listener can distinguish between the case when none of its neighbors transmit and the case when two or more of them transmit. In our model it is assumed that the listener cannot distinguish between the two cases (say, it hears noise in both cases). There is another common model in which it is assumed that the two cases are distinguishable (say, if no neighbor transmits, the listener hears silence, while if two or more neighbors transmit, the listener hears noise). A discussion justifying both models can be found in [Gal85, BGI92]. Willard [Wil86] studies a broadcast problem in a single multiaccess channel under this second model (i.e., when collision detection is available). He shows matching upper and lower bounds of $\Theta(\log \log n)$ expected time slots[2] in this model. Our main lemma implies an $\Omega(\log n)$ lower bound for the same problem in our model. Again, this lower bound holds even if the processors use different programs. Hence, we demonstrate a provable exponential gap between these two models.

The rest of this paper is organized as follows: section 2 contains some necessary definitions. Section 3 contains the proof of the main lemma in the *uniform* case, where all the processors use the same program. Section 4 contains the proof of the main lemma in the *nonuniform* case, where processors may use different programs. The

[2]Willard shows an $\Omega(\log \log n)$ lower bound in the single multiaccess channel model. Although this bound applies to a different model, it should be noted that his bound is also significantly restricted by the types of algorithms for which it applies. In particular, he requires independence between the decision whether to transmit in a certain time slot and the decisions made in previous time slots. In our case such a restriction is unacceptable, as the upper bound of [BGI92] has such dependencies. Also, he does not handle the case where each processor uses a different program.

proof for this case is based on a probabilistic reduction to the uniform case. Finally, in section 5, we prove the main theorem. The proof involves constructing a "difficult" network in a probabilistic way.

2. Preliminaries. A *radio network* is described by an undirected graph $G(V, E)$,[3] where $N = |V|$ and D is the diameter of the graph. The nodes of the graph represent processors of the network, and an edge between nodes v and u implies that v can send messages to u (and vice-versa). The *neighborhood* of a node u includes all the nodes v such that there is an edge (u, v) in E.

The time is viewed as divided into slots (or *rounds*). In any given slot, a node (processor) can either transmit some message (a string in $\{0,1\}^*$) or not (i.e., remain silent). A radio network has the property that if two or more nodes in the neighborhood of a node u transmit at the same time slot, then none of the messages is received at u. More formally, we can define the set of possible transmissions as $W = \{0,1\}^* \cup \{silent\}$. If exactly one of the node's neighbors transmits at time t and the message that this neighbor transmits is some $m \in \{0,1\}^*$, then m is received by the node. In any other case (i.e., if either none of the neighbors transmits or more than one neighbor transmits) this node hears *silent*. The *history* of length ℓ of a node is a vector in W^ℓ which consists of its view of the first ℓ rounds.

Each processor P_i in the radio network uses a *probabilistic* program. This program defines whether the processor will transmit at the next time slot j or not. As we are not concerned with the computational power of the processors we can simply view this program as a probability distribution, which may depend on the history. More formally, for each processor P_i and step j there is a probabilistic function $\Gamma_i^j : W^{j-1} \to W$ that, based on the history, determines the action of P_i in step j (i.e., whether it remains silent, or else the value of the message it sends). The *program* of P_i is a collection $\Gamma_i = (\Gamma_i^1, \Gamma_i^2, \ldots)$ that defines the actions of P_i in each step. A *protocol* $\mathcal{P}_{N,D}$ is simply a collection of N such programs, one per processor. A protocol is *uniform* if all processors use the same program. Otherwise, if each processor has a different program, the protocol is *nonuniform*. The above definition allows the protocols to use the values of N and D. On the other hand, the protocol "does not know" the topology of the graph, meaning that the same protocol must work for all graphs of N nodes and diameter D.

A *broadcast* protocol is a protocol that is initiated by a single processor, called *originator*, that holds a message M. Any other processor is inactive (i.e., it remains silent) until receiving a message for the first time. The aim of the protocol is that each processor in the network will receive a copy of the message M.

3. Uniform processors. In this section we prove the main lemma for the uniform case, where all processors use the same program. It shows that if there are n processors[4] arranged in a clique, then there exists a t $(2 \le t \le n)$ such that if t processors wish to transmit (we call these t processors the *participants*), then the expected number of rounds (time slots) until a round in which exactly one of them transmits is $\Omega(\log n)$. In fact, we show that this is the case for most of the t's of the form $t = 2^i$. Note that the assumption that the topology is not known to the processors, in the context of this lemma, means that t, the number of processors that

[3] None of the results presented in this work will be changed if the network is a directed one. However, it is common in this area to assume that the network is undirected.

[4] Note that we use here n (and not N) as the number of processors. This will be convenient while using the lemma in the proof of the theorem.

are trying to transmit, is not known to any processor. We can view the scenario as having a family of networks with $n+1$ nodes, composed from a clique of size n and an originator which is connected to t of the nodes in the clique. The (unknown) topology is chosen to be one of these networks.

For a broadcast protocol Π, we call a round *successful* if exactly one processor transmits. Let $E(T_\ell^\Pi)$ denote the expected number of rounds until the *first* successful round, given that the number of participants is 2^ℓ (the expectation is taken over the probabilistic choices of the processors).

LEMMA 1. *Let Π be a broadcast protocol, let the network be as above, and let n be an upper bound on the number of participants. Then,*

$$E_\ell[E(T_\ell^\Pi)] = \Omega(\log n),$$

where E_ℓ denotes the expectation when ℓ is chosen uniformly from the range $1 \le \ell \le \log n$.

Proof. The first observation that we make is that the lemma deals only with the *first* success. This, in a sense, allows us to get rid of the dependency in the history—we can assume that the (probabilistic) decision as to which rounds a processor tries to transmit is made at the beginning of the protocol. This is done by letting each of the 2^ℓ processors choose whether to transmit in round s or not in the same way as it chooses in the original protocol, when all previous rounds were unsuccessful. Clearly, as far as the *first* success is concerned, this modification has no effect on the protocol. Also, as only the first success is considered, it does not matter what the values of the messages that the processors try to transmit are. Hence, the decision of a processor on whether to transmit in round s may depend on the round number, s, and the probabilistic choices of the processor in the first $s-1$ rounds, but it does *not* depend on choices made by other processors.[5] Therefore, we can think about the processors as if they choose *in advance*, for every round $s = 1, 2, \ldots$, whether they will try to transmit.[6]

For simplicity of notation, we assume that n is a power of 2. Define

$$p_{s,\ell} \stackrel{\triangle}{=} Pr(\text{failure in rounds } 1, \ldots, s-1 \ \textbf{and} \ \text{success in round } s | 2^\ell \text{ participants}).$$

As the events described in the definition are disjoint (for *fixed* ℓ and different s's), and assuming that the protocol succeeds with probability 1 (no matter what ℓ is), we have for all ℓ

$$\sum_{s=1}^{\infty} p_{s,\ell} = 1. \tag{1}$$

At some point in the proof below, it will be inconvenient if $p_{s,\ell}$ depends on events that happen in previous rounds. However, we can get rid of this dependency simply by writing

$$p_{s,\ell} \le Pr(\text{ success in round } s | 2^\ell \text{ participants}). \tag{2}$$

[5]The message M that the processors need to broadcast also influences their decisions. However, it can be thought of as part of the program used by the processors.

[6]To avoid measurability concerns, it is convenient to assume that the protocol is such that s is in the range $1, \ldots, F$, for some *finite* F. If this is not the case, we can always choose F such that the probability of choosing only in the range $1, \ldots, F$ is arbitrarily close to 1. This will cause minor changes in our proof.

The next claim gives a bound on the sum of the success probabilities in a given round. Intuitively it says that you cannot have high probability of success in (a fixed) round s for more than a few values of 2^ℓ. This would imply that since the number of participants is unknown, $\Omega(\log n)$ rounds would be required to reach a success for all numbers of participants. Formally, we make the following claim.

CLAIM 2. *For any s,*

$$\sum_{\ell=1}^{\log n} Pr(\textit{success in round } s \mid 2^\ell \textit{ participants}) < 2.$$

Proof. Fix s. As already discussed, we assume that the processors make all their choices in advance. The *history* of choices of a processor is a string in $\{0,1\}^{s-1}$, where the value of the ith bit means trying ("1") or not trying ("0"). Define

$$q(s) \stackrel{\triangle}{=} Pr(\text{trying in round } s) = \sum_{\text{history } h} Pr(h) \cdot Pr(\text{trying in round } s|h).$$

Note that $q(s)$ does *not* depend on ℓ. We assume, without loss of generality, that $q(s) > 0$ (rounds with $q(s) = 0$ can be omitted from the protocol). Recall that a successful round is one in which exactly one processor is trying to transmit. Therefore,

$$Pr(\text{success in round } s \mid 2^\ell \text{ participants}) = 2^\ell \cdot q(s) \cdot (1-q(s))^{2^\ell - 1}.$$

We get

$$\begin{aligned}
\sum_{\ell=1}^{\log n} Pr(\text{success in round } s|2^\ell \text{ participants}) &= \sum_{\ell=1}^{\log n} 2^\ell q(s)(1-q(s))^{2^\ell-1} \\
&= q(s) \sum_{\ell=1}^{\log n} 2^\ell (1-q(s))^{2^\ell-1} \\
&\le 2 \cdot q(s) \sum_{j=1}^{n-1} (1-q(s))^j \\
&= 2 \cdot q(s) \cdot \frac{1-(1-q(s))^n}{q(s)} < 2
\end{aligned}$$

which completes the proof of the claim. □

Let k be a parameter (to be fixed later). We are interested in $\sum_{s=1}^{k} p_{s,\ell}$, which is intuitively the probability that, given that there are 2^ℓ participants, the algorithm succeeds in one of the first k rounds. Using equation (2) and Claim 2, we get

$$\sum_{\ell=1}^{\log n} \sum_{s=1}^{k} p_{s,\ell} \le \sum_{s=1}^{k} \sum_{\ell=1}^{\log n} Pr(\text{success in round } s|2^\ell \text{ participants}) < 2k. \tag{3}$$

By definition,

$$E_\ell[E(T_\ell^\Pi)] = \sum_{\ell=1}^{\log n} \frac{1}{\log n} \sum_{s=1}^{\infty} p_{s,\ell} \cdot s \ge \frac{k}{\log n} \sum_{\ell=1}^{\log n} \sum_{s=k}^{\infty} p_{s,\ell}.$$

By equation (1), this equals

$$\frac{k}{\log n}\sum_{\ell=1}^{\log n}\left(1-\sum_{s=1}^{k-1}p_{s,\ell}\right),$$

which, by equation (3), is greater than

$$\frac{k}{\log n}\cdot(\log n-2(k-1)).$$

By choosing $k=\frac{1}{4}\log n$, we have that

$$E_\ell[E(T_\ell^\Pi)]\geq\frac{1}{8}\log n+\frac{1}{2},$$

which completes the proof of the lemma.[7] □

4. Nonuniform processors. In this section we prove the main lemma for the nonuniform case, where the n processors may use *different* programs. The main idea of the proof is to "reduce" the nonuniform case to the uniform one, and use the result of the previous section (Lemma 1).

LEMMA 3. *Let* Π *be a protocol for* n *distinct processors* $P_1,\ldots,P_n$ *that run (possibly) different programs. Let* $E(T_\ell^\Pi)$ *denote the expected number of rounds until the* first *successful round, given that a* random *set of* 2^ℓ *processors participates (the expectation is taken over the choice of the set and the probabilistic choices made by the processors). Then*

$$E_\ell[E(T_\ell^\Pi)]=\Omega(\log n),$$

where ℓ *is chosen uniformly from the range* $1\leq\ell\leq\log n$.

Proof. As argued in the previous section, as only the first successful round is considered, each program can be thought of as a "schedule"—a choice of a subset of rounds in which the processor will transmit. Processor P_i chooses its schedule from a distribution μ_i.

We now define, based on the (possibly different) programs used by $P_1,\ldots,P_n$, a new program that will be used by each of L *uniform* processors $Q_1,\ldots,Q_L$: processor Q_j chooses (uniformly) at random $1\leq i\leq n$ and simulates the program of processor P_i. Namely, it chooses a schedule s with probability $\frac{1}{n}\sum_{i=1}^n\mu_i(s)$, where $\mu_i(s)$ is the probability that processor P_i chooses the schedule s. We denote by $c(Q_j)$ the processor P_i that Q_j chose to simulate. We emphasize that all the Q_j's run the *same* program (i.e., they are uniform), and that different Q_j's may choose to simulate the same processor P_i (we will choose L "small enough" so that this will happen only with a "small" probability).

The following claim says that, given that all the $c(Q_j)$'s are distinct for $Q_1,\ldots,Q_{2^\ell}$, then the probability distribution of the schedules chosen by the Q_j's is the same as that of a *random* set of 2^ℓ processors P_i.

CLAIM 4. *Let* $Q=\{Q_1,\ldots,Q_{2^\ell}\}$. *For every* $Q_j\in Q$, *let* $c(Q_j)$ *be a random processor* P_i. *If* $\forall j_1\neq j_2\ :\ c(Q_{j_1})\neq c(Q_{j_2})$, *then* $P=\{c(Q_j)|Q_j\in Q\}$ *is a random*

[7] In the original version of this paper [KM93], we proved a slightly better lower bound of $\frac{1}{4}\log n$; however, the proof here is simpler.

set of 2^ℓ processors (in $P_1, \ldots, P_n$), and the following holds: for every choice of 2^ℓ schedules $\vec{s}_{2^\ell} = (s_1, \ldots, s_{2^\ell})$,

$$Pr[\vec{s}_{2^\ell} | \textit{processors } Q \textit{ run }] = Pr[\vec{s}_{2^\ell} | \textit{processors } P \textit{ run }].$$

The following claim is the main tool in the reduction from the nonuniform case to the uniform case.

CLAIM 5. *Let Q be as above and let $Q' = \{Q'_1, \ldots, Q'_{2^\ell}\}$ be a set of 2^ℓ processors. Each processor Q'_j runs the program of Q_j at the odd steps and the* [BGI92] *program at the even steps. (Note that the* [BGI92] *program is also a uniform protocol, and therefore, so is the program run by the processors Q'.) Let β_ℓ be the probability that $\forall j_1 \neq j_2,\ c(Q_{j_1}) \neq c(Q_{j_2})$. Let $T_\ell^{Q'}$ be the random variable indicating the time of first success when the 2^ℓ identical programs in Q' run, and recall that T_ℓ^{Π} is the random variable indicating the time of first success when a random subset of 2^ℓ distinct programs $P_{i_1}, \ldots, P_{i_{2^\ell}}$ run. Then,*

$$E[T_\ell^{Q'}] \leq 2\beta_\ell E[T_\ell^{\Pi}] + 8(1-\beta_\ell)\log n.$$

In the above claim we mixed the given (unknown) protocol with the [BGI92] protocol. This is because we have no guarantee about the running time of the simulation, in the case when some Q_j's choose to simulate the same P_i. For example, a protocol that lets processor P_i transmit at time slot i would not terminate if all the Q_j simulate the same processor P_i.

Proof. Let *unique* be the event that $\forall Q_{j_1}, Q_{j_2} \in Q',\ c(Q_{j_1}) \neq c(Q_{j_2})$. Then,

$$E[T_\ell^{Q'}] = E[T_\ell^{Q'} | \textit{unique}] \cdot Pr[\textit{unique}] + \ E[T_\ell^{Q'} | \textit{not unique}] \cdot Pr[\textit{not unique}].$$

By definition, $Pr[\textit{unique}] = \beta_\ell$. By Claim 4,

$$E[T_\ell^{Q'} | \textit{unique}] \leq 2E[T_\ell^{\Pi}],$$

where the additional factor of 2 is due to the interleaving of the two protocols. In the case when the choices of $c(Q_j)$ are not unique, we cannot use the properties of the original protocol. However, we can use the fact that the [BGI92] protocol has the expected time until the first success of at most $4\log n$. Therefore,

$$E[T_\ell^{Q'} | \textit{not unique}] \leq 8\log n,$$

which completes the proof of the claim. □

The next claim says that with "high probability" the choices $c(Q_j)$ are unique.

CLAIM 6. *Let β_ℓ be the probability that $\forall j_1 \neq j_2,\ c(Q_{j_1}) \neq c(Q_{j_2})$, and assume that $2^\ell \leq n^{1/4}$. Then,*

$$\beta_\ell > 1 - \frac{1}{\sqrt{n}}.$$

Proof. Note that

$$Pr[j_1 \neq j_2 \text{ and } c(Q_{j_1}) = c(Q_{j_2})] = \frac{1}{n}.$$

Therefore,

$$\beta_\ell = Pr[\forall j_1 \neq j_2 \ : \ c(Q_{j_1}) \neq c(Q_{j_2})] \geq 1 - \binom{2^\ell}{2}\frac{1}{n}.$$

Since $2^\ell \leq n^{1/4}$ the lemma follows. □

Let $L = n^{1/4}$. By Claims 5 and 6,

$$E[T_\ell^{Q'}] \leq 2\beta_\ell E[T_\ell^\Pi] + (1-\beta_\ell)8\log n \leq 2E[T_\ell^\Pi] + \frac{8\log n}{\sqrt{n}}$$

or

$$E[T_\ell^\Pi] \geq \frac{1}{2}E[T_\ell^{Q'}] - \frac{4\log n}{\sqrt{n}}.$$

We now take the expectation over all values $1 \leq \ell \leq \log L$ and get

$$E_\ell[E[T_\ell^\Pi]] \geq \frac{1}{2}E_\ell[E[T_\ell^{Q'}]] - \frac{4\log n}{\sqrt{n}}.$$

By Lemma 1,

$$E_\ell[E[T_\ell^{Q'}]] = \Omega(\log L) = \Omega(\log n),$$

which implies that

$$E_\ell[E[T_\ell^\Pi]] = \Omega(\log n),$$

as desired. □

5. Main theorem. In this section we prove the main theorem. We show that for every broadcast algorithm that does not know the topology of the network, for every N and every D, there exist networks of N processors and diameter D such that the expected running time of the algorithm (until all processors receive the message) is $\Omega(D\log(N/D))$. This implies a similar lower bound for the *worst case* running time, when a small probability of error is allowed (which is the scenario in which the upper bound of [BGI92] is described).

Given an algorithm and the values N and D, we construct a network as follows. Let $n = N/D$, and assume for simplicity that n is a power of 2. We construct a complete layered network of $D+2$ layers. The first layer (layer 0) contains one node, s, which will be the originator of the broadcast. Each of the next D layers (layers $1, 2, \ldots, D$) consists of $n_i = 2^{\ell_i} \leq n$ nodes, where ℓ_i is chosen uniformly (and independently for each layer i) in the range $1, \ldots, \log n$. The last layer contains all the other nodes (so that the total number of nodes will be N). Each node in layer i is connected to all nodes in layers $i-1$ and $i+1$. (See Figure 1.)

Recall that the topology of the network is not known to the processors. (If the topology was known, then an efficient uniform protocol would be to let a processor at layer i broadcast with probability $1/n_i$, with expected time $O(D)$. A nonuniform protocol that knows the topology simply lets one node in each layer transmit.) The algorithm can depend, however, on other information that the processors have, in particular, the history, the number of steps, etc. (As mentioned, other information which is independent of the graph, such as the message M to be broadcast, the

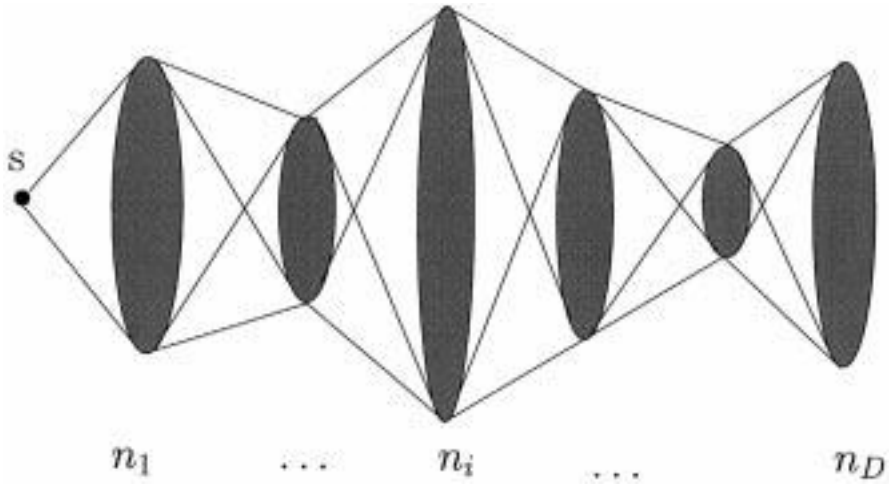


FIG. 1. *Structure of the network.*

processors' IDs, or the value of a clock, can be thought of as encoded into the programs of the processors.)

We discuss the uniform case, in the sense that all the processors at layer i have the same protocol. The extension to the nonuniform case employs the techniques of the previous section, and the proof is the same but the notation becomes cumbersome. (In particular, in the nonuniform case, at each layer i we will choose not only n_i but also a random set of n_i processors.) The main property of this construction is the following. For all i and all runs of the protocol, all the processors in layer i have the same view; every message received at one of these processors is received by all other processors at the same time. Therefore, the broadcast progresses in a layer-by-layer fashion. Moreover, this implies that all the processors in layer i choose schedules according to the same distribution μ (the choice of μ depends on the history, but all the processors of layer i share the same history), which allows us to use Lemma 1.

Finally, before going into the details, we make one more assumption that makes our argument simpler. We give the processors of layer i, at the time they get the first message from a processor in layer $i-1$, all the other messages they will get from layer $i-1$ in the future, as well as the actual values of $\ell_1, \ldots, \ell_{i-1}$. As this extra information can only help the processors to make the broadcast faster, we are allowed to make this assumption.

Let t_i be the random variable indicating the number of rounds from the time the processors of layer i get the message (and become active) until their success (the first time that a single processor in layer i transmits). We need to show that for some choice of $\ell_1, \ldots, \ell_D$ we get $E_\Pi(\sum_{i=1}^{D} t_i) = \Omega(D \log(N/D))$, where the expectation is taken over the random choices of the algorithm Π. Certainly, it is enough to show that $E_{\ell_1,\ldots,\ell_D,\Pi}(\sum_{i=1}^{D} t_i) = \Omega(D \log(N/D))$. By linearity of expectation, we get

$$E_{\ell_1,\ldots,\ell_D,\Pi}\left(\sum_{i=1}^{D} t_i\right) = \sum_{i=1}^{D} E_{\ell_1,\ldots,\ell_D,\Pi}(t_i).$$

So all we have to bound now is $E_{\ell_1,\ldots,\ell_D,\Pi}(t_i)$. Clearly, the choice of $\ell_{i+1}, \ldots, \ell_D$ has

no influence on the expectation of t_i; i.e.,

$$E_{\ell_1,\ldots,\ell_D,\Pi}(t_i) = E_{\ell_1,\ldots,\ell_i,\Pi}(t_i).$$

Also, by the discussion above, with every history (which depends on the random choices made in the first $i-1$ layers, including the choice of $\ell_1,\ldots,\ell_{i-1}$) we can associate a probability distribution μ used by the processors in layer i to choose their schedules. (Note that since we assume that the processors of layer i get all the future information with the first message, they can make all their random choices at this time.) Therefore, we can write

$$E_{\ell_1,\ldots,\ell_i,\Pi}(t_i) = \sum_{b_1,\ldots,b_{i-1}} E_{\ell_i,\Pi}(t_i|\ell_1 = b_1,\ldots,\ell_{i-1} = b_{i-1})\cdot Pr[\ell_1 = b_1,\ldots,\ell_{i-1} = b_{i-1}]. \tag{4}$$

It remains to bound the expression $E_{\ell_i,\Pi}(t_i|\ell_1 = b_1,\ldots,\ell_{i-1} = b_{i-1})$. As mentioned, we allow the processors at layer i to have access to $b_1,\ldots,b_{i-1}$ (the actual values of $\ell_1,\ldots,\ell_{i-1}$). Therefore, we need to evaluate $E_{\ell_i,\Pi_i}(t_i)$, where Π_i is the protocol at layer i, with the additional information about the lower layers. By Lemma 1, for each such Π_i,

$$E_{\ell_i,\Pi_i}(t_i) \geq c\log n$$

for some constant c. Therefore, for every $b_1,\ldots,b_{i-1}$, we have

$$E_{\ell_i,\Pi}(t_i|\ell_1 = b_1,\ldots,\ell_{i-1} = b_{i-1}) \geq c\log n,$$

which by (4), implies

$$E_{\ell_1,\ldots,\ell_i,\Pi}(t_i) \geq c\log n.$$

This implies

$$E_{\ell_1,\ldots,\ell_D,\Pi}\left(\sum_{i=1}^{D} t_i\right) = \Omega(D\log n) = \Omega(D\log(N/D)),$$

which completes the proof of our main theorem.

THEOREM 7. *For any nonuniform broadcast protocol, for every number of processors N and every diameter D, there exists a network in which the expected time to complete a broadcast is $\Omega(D\log(N/D))$.*

When $D \leq N^{1-\varepsilon}$, the above proof shows a lower bound of $\Omega(D\log N)$. Combining our result with the results of Alon et al. [ABLP91] and Bar-Yehuda, Goldreich, and Itai [BGI92], we have the following tight result.

COROLLARY 8. *For any nonuniform broadcast protocol, for every number of processors N and every diameter D, there exists a network in which the time to complete a broadcast is $\Omega(\log^2 N + D\log(N/D))$. Furthermore, there is a (uniform) protocol that requires only $O(\log^2 N + D\log N)$ expected time (which is tight for $D \leq N^{1-\varepsilon}$).*

Note that unlike [ABLP91] we show that for any protocol there exists a network for which the lower bound holds, while they prove that there exists a network on which any protocol requires the lower bound.

Acknowledgments. We wish to thank Oded Goldreich and the anonymous referees for their very useful comments.

REFERENCES

[ABLP91] N. Alon, A. Bar-Noy, N. Linial, and D. Peleg, *A lower bound for radio broadcast*, J. Comput. System Sci., 43 (1991), pp. 290–298.

[ABLP92] N. Alon, A. Bar-Noy, N. Linial, and D. Peleg, *Single round simulation on radio networks*, J. Algorithms, 13 (1992), pp. 188–210.

[BGI91] R. Bar-Yehuda, O. Goldreich, and A. Itai, *Efficient emulation of single-hop radio network with collision detection on multi-hop radio network with no collision detection*, Distrib. Comput., 5 (1991), pp. 67–71.

[BGI92] R. Bar-Yehuda, O. Goldreich, and A. Itai, *On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization*, J. Comput. System Sci., 45 (1992), pp. 104–126.

[BII93] R. Bar-Yehuda, A. Israeli, and A. Itai, *Multiple communication in multi-hop radio networks*, SIAM J. Comput., 22 (1993), pp. 875–887.

[CK85a] I. Chlamtac and S. Kutten, *On broadcasting in radio networks—problem analysis and protocol design*, IEEE Trans. Comm., COM-33 (1985), pp. 1240–1246.

[CK85b] I. Chlamtac and S. Kutten, *A spatial reuse TDMA/FDMA for mobile multi-hop radio networks*, in INFOCOM, 1985, pp. 389–394.

[CK87] I. Chlamtac and S. Kutten, *Tree-based broadcasting in multihop radio networks*, IEEECOM, C-36 (1987), pp. 1209–1223.

[CW87] I. Chlamtac and O. Weinstein, *The wave expansion approach to broadcasting in multihop radio networks*, in INFOCOM, pp. 874–881, 1987.

[Gal85] R. Gallager, *A perspective on multiaccess channels*, IEEE Trans. Inform. Theory, 31 (1985), pp. 124–142.

[GM95] I. Gaber and Y. Mansour, *Broadcast in radio networks*, in Proc. 6th ACM-SIAM Symposium on Discrete Algorithms, SIAM, 1995, pp. 577–585.

[KM93] E. Kushilevitz and Y. Mansour, *An $\Omega(D \log(N/D))$ lower bound for broadcast in radio networks*, in Proc. 12th ACM Symp. on Principles of Distributed Computing, 1993, pp. 65–74.

[Tan81] A. S. Tanenbaum, *Computer Networks*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[Wil86] D. E. Willard, *Log-logarithmic selection resolution protocols in a multiple access channel*, SIAM J. Comput., 15 (1986), pp. 468–477.

SIAM J. COMPUT.
Vol. 27, No. 3, pp. 713–736, June 1998
© 1998 Society for Industrial and Applied Mathematics
009

OPTIMAL ON-LINE SEARCH AND SUBLINEAR TIME UPDATE IN STRING MATCHING*

PAOLO FERRAGINA† AND ROBERTO GROSSI‡

Abstract. This paper investigates the problem of searching on-line for the occurrences (*occ*) of an arbitrary pattern of length p in a text of length n subjected to some updates after its preprocessing. Each text update consists of inserting or deleting an arbitrary string of length y. We present the first dynamic algorithm that achieves *optimal* query time, i.e., $\Theta(p+occ)$, *sublinear* time per update, i.e., $O(\sqrt{n}+y)$, and *optimal* space, i.e., $\Theta(n)$, in the worst case. As a result, our algorithm obtains the same query time and space usage of suffix trees [McCreight, *J. Assoc. Comput. Mach.*, 23 (1976), pp. 262–272] while improving their $O(n+y)$ update performance.

Key words. string matching, suffix tree, text indexing, dynamic data structures

AMS subject classifications. 68P05, 68P20, 68Q20, 68Q25

PII. S0097539795286119

1. Introduction.

The problem of encoding all the substrings of a given string for string-matching purposes is amply treated in current literature. Given a *text* string $T[1,n]$ made up of n characters taken from an ordered alphabet Σ, we want to *preprocess* T in order to represent all of its substrings compactly (there could be as many as $\Theta(n^2)$) and therefore answer the following two *on-line* search queries efficiently for any *pattern* string $P[1,p]$:

(a) check to see if P is one of T's substrings;

(b) find all of T's substrings equal to P (let *occ* be their number).

There are three main groups of methods used for preprocessing T and solving the above problem. Over the last twenty years, they have been "rediscovered" in scientific literature disguised under different names and studied in various forms (e.g., see [3, 9]). The following is a brief list of them (we assume that Σ has constant size).

(1) The first group is made up of *compacted tries* that include, among other things, the compacted prefix bi-tree [33] (or prefix tree [9]), the PAT tree [19], the position tree [1, 23, 26], the repetition finder [30], the subword tree [3, 9], and the suffix tree [28] (the latter is definitely the best known). The text suffixes are stored in the compacted trie leaves so that every text substring is represented by a (unique) path descending from the root. Two examples of suffix trees are given in Fig. 1. Compacted tries require optimal $O(n)$ space and construction time (e.g., see [28, 33]). A query can be answered in optimal time, namely in $O(p)$ time for query (a) and in $O(p+occ)$ time for query (b). It is also possible to obtain some efficient statistics on T's substrings (e.g., see [3, 33]). Furthermore, compacted tries have successfully been generalized to matrices [4, 17, 18] and parameterized strings [5].

(2) The second group is made up of *automata* or *word graphs*, such as the complete inverted file [7], the directed acyclic word graph (DAWG) [6], and the minimal suffix and factor automata [10, 11]. They are either labeled directed acyclic graphs

*Received by the editors May 17, 1995; accepted for publication (in revised form) April 5, 1996. This research was supported by MURST of Italy. A preliminary version of this paper was presented at the IEEE Symposium on Foundations of Computer Science, 1995.

http://www.siam.org/journals/sicomp/27-3/28611.html

†Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa, Italy (ferragin@di.unipi.it).

‡Dipartimento di Sistemi e Informatica, Università di Firenze, via Lombroso 6/17, 50134 Firenze, Italy (grossi@dsi2.dsi.unifi.it).

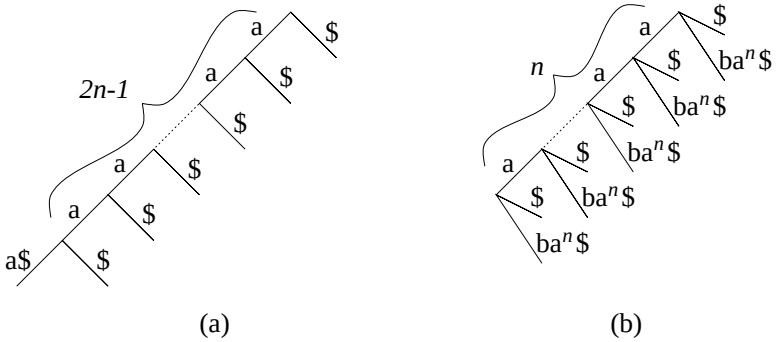


FIG. 1. *The suffix tree for* (a) $T = a^{2n}$ *and* (b) $T' = a^n b a^n$.

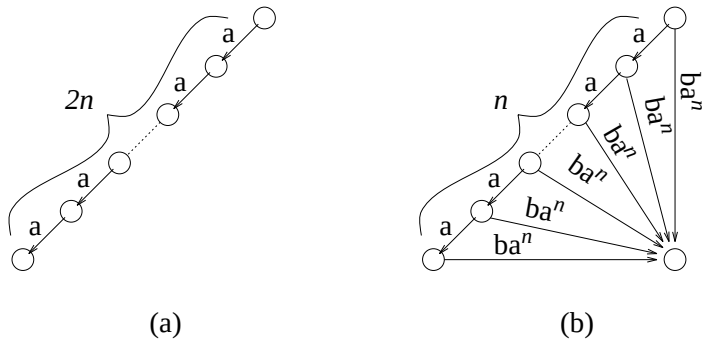


FIG. 2. *The compact DAWG for* (a) $T = a^{2n}$ *and* (b) $T' = a^n b a^n$.

or automata whose nodes correspond to the nodes of the compacted trie built on the string T [7] or T's reversal [6, 10, 11]. Two examples of compact DAWGs are given in Fig. 2. Their space usage, construction time, and query time are identical to the ones required by compacted tries. Directed acyclic word graphs and minimal suffix and factor automata allow us to perform queries (a), whereas complete inverted files enable more queries than just (a) and (b).

(3) Finally, the third group is characterized by *lexicographically ordered data structures*, such as the suffix array [27] and the dynamic suffix array [14], the PAT array [19], and the SB-tree [15]. They all maintain T's suffixes in lexicographic order, which can be equivalently obtained by traversing the compacted trie leaves. Two examples of suffix arrays in their simplest form are given in Fig. 3. The space required is still optimal, i.e., $O(n)$, but the construction time is $O(n \log n)$ in the worst case (which becomes $O(n)$ on the average for suffix arrays [27]). Query (a) takes $O(p+\log n)$ time and query (b) takes $O(p + \log n + occ)$ time (see [27] for further details).

In this paper, we only discuss query (b) because query (a) derives from it. As previously mentioned, groups (1)–(3) are very efficient and, when dealing with a *static* text, there is no better asymptotic bound possible than (1) and (2). However, in many common situations—e.g., text editors and text retrieval systems [32, section 5.3]—the preprocessed text T undergoes some changes. Some special cases can be handled by the methods in (1)–(3), namely, strings can be appended to one end of T with suffix trees [2, 21, 33] and compact DAWGs [7], or a subset of T's suffixes can be *logically*

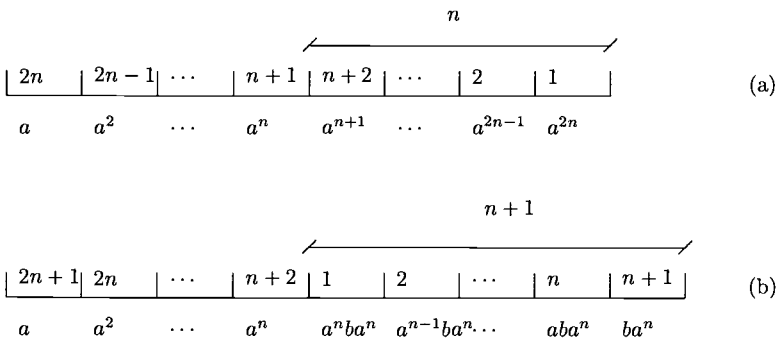


FIG. 3. *The suffix array for* (a) $T = a^{2n}$ *and* (b) $T' = a^n b a^n$.

deleted and undeleted [14]. However, the general problem of *dynamically* modifying T in arbitrary positions by inserting or deleting *strings* after preprocessing, seems a difficult task to solve efficiently. Evidence of this can be found in the following illustrative example: let us assume that $T = a^{2n}$ becomes $T' = a^n b a^n$ after a single character b is inserted into it, with $a \neq b$. Then $\Omega(n)$ locations must be updated in each of the known data structures that implement (1)–(3) (see Figs. 1–3). Consequently, updating the data structures basically costs the same as rebuilding them from scratch! Furthermore, as pointed out in [20], a naive method for avoiding the rebuilding that consists of keeping a "to-do-list" of changes is still very expensive.

Due to the trivial $\Omega(y)$ lower bound to the time needed for inserting or deleting a string $Y[1, y]$, we realized the importance of obtaining a *sublinear* time update, i.e., $o(n) + \Theta(y)$ for *all* y, while maintaining optimal query time, i.e., $O(p + occ)$, and optimal space usage, i.e., $O(n)$. In this paper we illustrate how to solve this problem.

In current literature, an important step has been done with *dynamic text indexing*, introduced in [20] with the aim of attaining an efficient time trade-off between pattern searching and single-character (text) updating. This trade-off was generalized in [13, 14] to text *string* insertions (or deletions) which are seen as atomic changes rather than sequences of single-character changes. It is worth noting that all of these dynamic text indexing results deal with *nonoblivious* queries: while the length of the "history" of insertions and deletions increases, the query's performance degenerates independently of the current text and pattern length. It is therefore reasonable to design *oblivious* dynamic text indexing algorithms whose query time does not explicitly depend on the "history." There are some well-known examples of oblivious dynamic algorithms: dynamic graph algorithms, for example, have a time complexity that only depends on the current graph size (e.g., see [12]).

We start out by proposing a simple technique for obtaining oblivious queries by means of a text partitioning technique. We provide the first dynamic text indexing algorithm that achieves an *optimal* query time, i.e., $O(p + occ)$. Updating the text under the insertion (or deletion) of a string $Y[1, y]$ takes $O((y + \sqrt{n \log n}) \log(n + y))$ time. The space required is $O(n \log n)$ (see Theorem 3.2).

However, our main contribution in this paper is presenting the first dynamic algorithm that achieves in the worst case:

- optimal query time $\Theta(p + pocc)$ (Theorem 4.3);
- sublinear time per update $O(\sqrt{n} + y)$ (Theorem 4.4);
- optimal space $\Theta(n)$.

As far as we know, this is the first dynamic algorithm that provides both a query in optimal $\Theta(p + occ)$ time and an update in sublinear $o(n) + \Theta(y)$ time for *every* y, in optimal space. Our result is significant for the following reasons: "start-over" solutions requiring $O(n + y)$ time per update can be obtained by running the static algorithms from scratch. These start-over solutions show that the known dynamic text indexing algorithms still obtain a trade-off because their fast updates are *not always* $o(n)+\Theta(y)$ *for every* y (e.g., let us take $y = \Omega(\frac{n}{\log n})$ in Theorem 3.2). Moreover, their space usage is $O(n \log n)$ (except for [20]). Therefore, if we consider the start-over solutions from a theoretical point of view, to our surprise, we discover that they are the best-known text updating methods for *almost all* input strings (see footnote[1]). In our results, we do not simply reduce the update cost and the amount of space required by a polylogarithmic factor (still maintaining an optimal query time), but we also show that our oblivious algorithm is *not* a trade-off because it is *always better* than the start-over solutions for every y. Since no better asymptotic bounds can exist for $y = \Omega(\sqrt{n})$, our algorithm is the best one possible in this case. Furthermore, our solution is still *optimal* for a *static* text T. As a result, we obtain the same query time and space usage of suffix trees, automata, etc., while improving their update performance.

Our paper is organized as follows. We discuss some preliminary notions in section 2. In section 3 we introduce the notion of "balanced" text partitioning, while we outline our main result in section 4. In sections 5–7, we describe the implementation details and new techniques.

2. Preliminaries. Our preliminary remarks regard string periodicity [24] and suffix trees [28, 33]. (Readers familiar with them can skip this section.)

Let $X[1, m]$ be a string of m characters taken from the ordered alphabet Σ, and let $X[1, i]$ be the ith prefix and $X[j, m]$ be the jth suffix of X. *A period* of X is any proper prefix π such that X is in the form of $\pi^r\pi'$, where $r \geq 1$ and π' is a (maybe empty) prefix of π. Analogously, $X[i] = X[i + |\pi|]$ for every $1 \leq i \leq m - |\pi| + 1$. *The* period π of X is its *shortest* period. The following lemma is well known.

LEMMA 2.1 (see Knuth, Morris, and Pratt (KMP) [24]). *If p and q are the lengths of any two periods of X, such that $p+q \leq m+gcd(p, q)$, then $gcd(p, q)$ is also the length of a period of X.*

The following simple fact regarding X's periodicity will be useful later.

FACT 2.2. *Any substring $X[i, j]$ longer than $|\pi|$ has a cyclic shift of π's characters as a period.*

We wish to point out that a cyclic shift of π is not necessarily *the* period of $X[i, j]$ (which may be shorter).

For $1 \leq i \leq m$, the decomposition of each prefix $X[1, i]$ in the form of $\pi_i^r\pi_i'$, where π_i is *the* period of $X[1, i]$, can be obtained in $O(m)$ time with KMP's algorithm [24]. The same computation can be performed for each of X's suffixes by using X's reversal, denoted by X^R. We will also use the notion of border. A *border* is a proper prefix of X that is also its suffix. We can compute the *longest* border *Shift*$(X[1, i])$ of each prefix $X[1, i]$ in a total of $O(m)$ time by using the algorithm in [24].

[1] For example, we want to delete a generic string $Y[1, y]$ and let us assume it costs $O(y \log n)+o(n)$ time. This bound is better than $O(n + y)$ when $y = O(n/\log n)$. This happens for $\Theta(|\Sigma|^{cn/\log n})$ input strings Y, for some constant $c > 0$, whereas for other $\Theta(|\Sigma|^n - |\Sigma|^{cn/\log n})$ input strings Y the start-over solutions in (1)–(2) are faster. In this case, the former term is asymptotically smaller than the latter.

The *suffix tree* ST_X [28, 33] for the string X is a compacted trie built on the augmented string $X[1, m+1]$, where $X[m+1] = \$ \notin \Sigma$ is an endmarker. It stores the suffixes $X[i, m+1]$ in its leaves, for $1 \leq i \leq m$, and can be built optimally in $O(m)$ time and space (e.g., see [28]). Each ST_X arc is labeled with a substring $X[i, j]$ represented by a triple (X, i, j), and sibling arcs are ordered according to the first character of their label (see Fig. 1). We adopt the following terminology for suffix trees. Given a node $u \in ST_X$, the concatenation of the labels encountered along the downward path in ST_X from the root to u is denoted by $W(u)$. This node u is called the *locus* of $W(u)$. The *extended locus* of a string Y in ST_X is the unique node v, if it exists, such that Y is a prefix of $W(v)$ and $W(p(v))$ is a proper prefix of Y, where $p(v)$ is v's parent in ST_X. The following property of ST_X is used throughout the paper, and ensures that, given a node $u \in ST_X$, all the leaves descending from u store the suffixes having the same prefix $W(u)$.

PROPERTY 2.3. *Two suffixes have a common prefix, i.e. Y, if and only if they share a path from the root to the extended locus of Y.*

We can preprocess ST_X in $O(m)$ time [22, 31] in order to answer constant-time queries for finding the *lowest common ancestor* of any two nodes (hereafter called *LCA query*). LCA queries can be used for finding the *longest common prefix* of any two substrings of X, in $O(1)$ time, by Property 2.3 (see [25]).

We call the suffix tree generalization to a set Δ of strings [2, 21] a *generalized suffix tree* (GST_Δ for short). It is the compacted trie obtained by "superimposing" suffix trees ST_X for all $X \in \Delta$ incrementally, in $O(|X|)$ time each. Two arcs are superimposed whenever their labels have a common prefix. Multiple equal strings are associated with the same leaf and kept in a doubly linked list. The fundamental point is that, given $\Delta = \{X_1, X_2, \ldots, X_k\}$, the compacted trie GST_Δ is isomorphic to $ST_{X_1\$X_2\$\ldots\$X_k\$}$, where each instance of \$ is assumed to be different from the others. The out-degree of each node can be kept bounded by alphabet size $|\Sigma|$ according to [2]. The insertion of a suffix determines a leaf insertion (and sometimes the insertion of its parent). The symmetrical operation of deleting a string X from Δ can also be obtained in $O(|X|)$ time. In this case, too, Property 2.3 holds. See [2, 21] for further details about GST_Δ building and updating by means of McCreight's algorithm.

3. A simple technique for optimal query: "Balanced" partition. In this section, we introduce a simple technique for maintaining a "balanced" partition of text T; that is, we decompose T into substrings of roughly the same size. We give an example of its application to a known dynamic text indexing solution and obtain the first *oblivious* algorithm having *optimal* query time. We use the following result.

THEOREM 3.1 (see Ferragina and Grossi [14, section 6.4]). *The dynamic text indexing problem on $T[1, n]$ can be solved with the following bounds: finding the occurrences of $P[1, p]$ requires $O(p + occ + |\mathcal{L}| \log p + \log n)$ time, where occ is the number of occurrences; inserting or deleting a string $Y[1, y]$ takes $O(y \log(y + n))$ time. The space required is $O(n \log n)$.*

Our idea for using this result is based on maintaining $T = I_1 \cdots I_k$ as a list $\mathcal{L} = \{I_1, \ldots, I_k\}$ of its substrings (called *intervals*) whose length satisfies $\ell \leq |I_i| < 2\ell$, where ℓ is a parameter specified later on. As a result, $\mathcal{L}$'s size is $k = \Theta(\frac{n}{\ell})$ and this is called a *balanced* partition. Let us maintain an $O(n)$-size generalized suffix tree GST_Δ built on the set $\Delta = \{I_1 I_2, I_2 I_3, \ldots, I_{k-1} I_k\}$ of pairs of consecutive intervals. Any two consecutive strings in Δ overlap for exactly one of $\mathcal{L}$'s intervals.

We can now find P in two main cases according to its length p: (1) If $p \leq \ell$, then each occurrence is entirely contained in a pair of consecutive intervals. Therefore,

all the occurrences are easily found in $O(p + occ)$ time by accessing the leaves that descend from P's extended locus into GST_Δ (by Property 2.3). Multiple occurrences are listed twice at most. (2) If $p > \ell$, then listing all the occurrences takes $O(p+occ+\frac{n}{\ell}\log p + \log n)$ time by Theorem 3.1 (since $|\mathcal{L}| = O(\frac{n}{\ell})$). By fixing $\ell = \sqrt{n \log n}$, the query time becomes optimal, i.e., $O(p + occ)$.

Text partitioning can be maintained consistently under string insertions and deletions by means of some split and merge operations on intervals. Let us assume that we must insert a string $Y[1, y]$ into an interval $I_i = I'I''$ to produce the string $X = I'YI''$. We split X into $h = \lfloor \frac{|X|}{\ell} \rfloor$ intervals $I'_1, \ldots, I'_h$, such that $|I'_1| = \cdots = |I'_{h-1}| = \ell$ and $\ell \leq |I_h| < 2\ell$. Text T must be updated by replacing I_i with the intervals created because of X. We also update GST_Δ consistently by deleting the strings $I_{i-1}I_i$ and I_iI_{i+1} and inserting the strings $I_{i-1}I'_1, I'_1I'_2, \ldots, I'_hI_{i+1}$ of length $O(\ell)$ each. Since there are $O(y/\ell + 1)$ of them, the update requires $O(y + \ell)$ total time by the algorithms in [2]. We then update the data structures in Theorem 3.1 by deleting the whole I_i and inserting the sequence of $O(y/\ell + 1)$ strings $I'_1, \ldots, I'_h$ into $\mathcal{L}$, in $O((y + \ell)\log(n + y + \ell))$ total time. Since we fixed $\ell = \sqrt{n \log n}$, the bound becomes $O((y + \sqrt{n \log n}) \log(y + n))$.

When deleting a string $Y[1, y]$, we remove $\mathcal{L}$'s shortest sublist that denotes T's substring $I'YI''$ entirely containing Y. Then we insert string $I'I''$ into T again, with $|I'I''| < 4\ell$. This requires making a deletion in $O((y + \ell)\log n)$ time and an insertion of no more than two new intervals in $O(\ell \log(n + \ell))$ time (by Theorem 3.1). Since we fixed $\ell = \sqrt{n \log n}$, the bound becomes $O((y + \sqrt{n \log n}) \log n)$. Our first result is therefore proved.

THEOREM 3.2. *Searching for a pattern $P[1, p]$ in the text $T[1, n]$ requires optimal $O(p+occ)$ time, where occ is the total number of occurrences. Updating the text under the insertion or deletion of a string $Y[1, y]$ takes $O((y + \sqrt{n \log n}) \log(y + n))$ time. The space required is $O(n \log n)$.*

4. Optimal query, sublinear time update, and optimal space. We now give a high-level overview of our main result for finding pattern occurrences optimally, i.e., $O(p+occ)$ time, while also obtaining sublinear time update, i.e., $O(\sqrt{n}+y)$, and optimal space, i.e., $O(n)$. We combine the basic idea of balanced partitioning described in section 3 with some new techniques for updating and searching efficiently in the intervals' list $\mathcal{L}$. In the following, we indicate the ideas we base our data structures and algorithmic techniques on.

We recall that the text is maintained as a partition of intervals whose length goes from ℓ to 2ℓ and which are stored into a list $\mathcal{L} = \{I_1, I_2, \ldots, I_k\}$ of size $k = \Theta(\frac{n}{\ell})$. String insertions and deletions are implemented by performing split and merge operations on $\mathcal{L}$'s intervals (to maintain their length between ℓ and 2ℓ). By setting $\ell = \Theta(\sqrt{n})$, we obtain a balanced partition in which an interval's length is proportional to the intervals' number. Due to this partition the pattern can be searched for in two different ways depending on its length. We call a pattern *short* if it is not longer than $c\ell$ (for a proper constant c, to be fixed below) and call it *long* otherwise.

Short patterns can be searched for by maintaining a generalized suffix tree GST_S on a set S of text substrings obtained by concatenating a sequence of consecutive intervals. For each interval I_i, we insert the smallest substring $I_i \ldots I_{i+r}$ whose length is at least $c\ell + 4\ell$ into GST_S (which contains $O(\frac{n}{\ell})$ strings). This makes sure that each short pattern occurrence is entirely contained in a string stored in GST_S because no interval is longer than 2ℓ.

Long patterns can be searched for in a more involved way. Since $p = \Omega(\ell) = \Omega(\frac{n}{\ell})$,

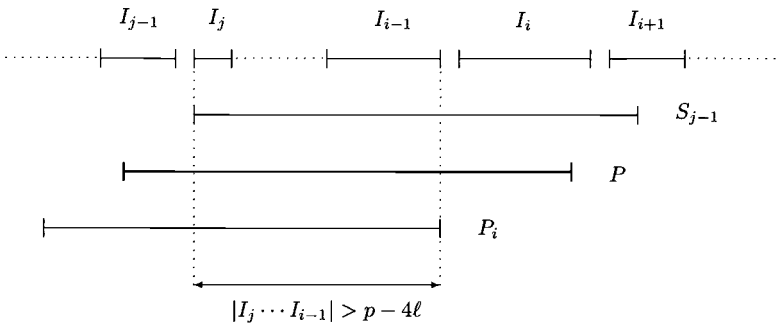


FIG. 4. *A pattern occurrence ending in interval I_i.*

we have enough time for scanning $\mathcal{L}$. Therefore, we process this list and, for each interval I_i, we compute both the pattern's longest prefix P_i that is a suffix of $I_1 \dots I_{i-1}$ and the pattern's longest suffix S_i that is a prefix of $I_{i+1} \dots I_k$. After that, we put P_i's and S_i's together to find the pattern occurrences. Our approach is different from the ones in [13, 14, 20] because it exploits long-pattern periodicity. We now want to state some constants that we will use later on in our algorithms and data structures.

Let us assume that there are some occurrences ending in I_i. Since the pattern is long, we have the property that each such occurrence completely overlaps a sequence of consecutive intervals preceding I_i. Let us choose the minimum integer j, such that the sequence $I_j I_{j+1} \dots I_{i-1}$ is maximal and contains all of the occurrences ending in I_i. In the worst case, an occurrence ends in I_i's last position and occupies $|I_i| < 2\ell$ positions to the right of I_{i-1} and less than 2ℓ positions to the left of I_j. We therefore have (see Fig. 4):

$$|I_j I_{j+1} \dots I_{i-1}| > p - 4\ell. \tag{4.1}$$

It is worth noting that P_i and S_{j-1} actually cover a text's substring containing all the occurrences ending in I_i, and $I_j I_{j+1} \dots I_{i-1}$ is the overlapping part between P_i and S_{j-1}. We use P_i and S_{j-1} to determine these occurrences. Since we shall prove that the sum of P_i's and S_{j-1}'s period lengths is upper bounded by 6ℓ, we want to have $p - 4\ell > 6\ell$ in (1). This makes sure that $|I_j I_{j+1} \dots I_{i-1}| > 6\ell$, and thus both P_i's and S_{j-1}'s periods occur in $I_j I_{j+1} \dots I_{i-1}$. We then apply Lemma 2.1 and find out that these periods are equal; because of this, we are able to determine all the occurrences in optimal time. Therefore, in order to guarantee that $p - 4\ell > 6\ell$ in (1), we define a long pattern to be of length $p > 10\ell$ (and so we set $c = 10$). In brief, a pattern is short if $p \leq 10\ell$, and GST_S stores strings at least $(c+4)\ell = 14\ell$ long. We work with P_i's and S_{j-1}'s periods whose total length is no more than 6ℓ. We use these constants to introduce our main data structures.

4.1. A data structure pool. We let $\mathcal{L} = \{I_1, \dots, I_k\}$, where $k = \Theta(\frac{n}{\ell})$ and $\ell \leq |I_i| < 2\ell$. There are also two dummy intervals $I_0 = I_{k+1}$ of length 14ℓ. Both I_0 and I_{k+1} are entirely made up of \$'s, where \$ does not occur in Σ. We define for each interval $I_i \in \mathcal{L}$:

- $string(I_i) = I_i \cdots I_{i+r}$, where r is the smallest integer such that $|I_i| + \cdots + |I_{i+r}| \geq 14\ell$ (we have $r \leq 14$).
- C_i is a set of two strings, called I_i's *cover*, which is made up of I_i's prefix and suffix of length ℓ. Since $\ell \leq |I_i| < 2\ell$, the two strings are *overlapping*.

- L_i is the suffix of $I_0 I_1 \cdots I_i$ of *length* 6ℓ and R_i is the prefix of $I_i \cdots I_k I_{k+1}$ of *length* 6ℓ.

Our data structure pool is made up of four main data structures $\{GST_S, GST_C, GST_L, GST_R\}$ and is maintained together with list $\mathcal{L}$. Let $P[1,p]$ be an arbitrary pattern to be queried in the pool. We define:

- GST_S is a generalized suffix tree for set $\Delta_S = \{string(I_1), \ldots, string(I_k)\}$. We perform the on-line search of short patterns in the strings of Δ_S by finding the pattern's extended locus in GST_S.
- GST_C is a generalized suffix tree for set $\Delta_C = \cup_{i=1}^{k} C_i$ (where each C_i is a set of two strings of the same length ℓ). We can perform on this set:
  - `Query-C(`P`)`: For all $X \in \Delta_C$, find a pattern substring (if it exists) equal to X, denoted by $SubEq(P, X)$.
- GST_R and GST_L are two *augmented* generalized suffix trees built on the strings in the set $\Delta_R = \{R_1, \ldots, R_k\}$ and on the *reversal* of the strings in the set $\Delta_L = \{L_1, \ldots, L_k\}$, respectively. We can perform on these sets:
  - `Query-R(`P`)`: For all $X \in \Delta_R$, find the pattern's *longest suffix* that is a *prefix* of X, denoted by $SufPref(P, X)$.
  - `Query-L(`P`)`: For all $X \in \Delta_L$, find the pattern's *longest prefix* that is a *suffix* of X, denoted by $PrefSuf(P, X)$.

Each data structure in the pool has $O(n)$ size and $O(\ell)$ depth, and the pool requires a total of optimal $O(n)$ space. It is not straightforward to implement the above queries because of the changes that can be performed on the GSTs in the pool by means of the following operations:

- Given a set $\Delta \in \{\Delta_S, \Delta_C, \Delta_L, \Delta_R\}$ and a string $Z[1,z]$, with $z = O(\ell)$, the operation `Add(`Z, Δ`)` (resp., `Remove(`Z, Δ`)`) adds string Z to (resp., removes it from) Δ and updates GST_Δ.

In section 5, we give a detailed description of the data structure pool along with a complexity analysis of their query and update operations.

4.2. Pattern searching. The algorithm listing all the occurrences (*occ*) of a pattern $P[1,p]$ in the current text T is outlined below, while a technical discussion of it can be found in section 6.

ALGORITHM `Find(`P`)`.

(1) If the pattern is short ($p \leq 10\ell$), then we find its extended locus in GST_S and list the distinct occurrences associated with the leaves descending from the locus. Otherwise the pattern is long ($p > 10\ell$) and we go on to steps (2) and (3).
(2) For all $i = 1, \ldots, k$, we find the pattern's *longest prefix* P_i that is a suffix of $I_1 \cdots I_{i-1}$ and the pattern's *longest suffix* S_i that is a prefix of $I_{i+1} \cdots I_k$. We proceed inductively by scanning list $\mathcal{L}$ from left to right for P_i and from right to left for S_i. We exploit the data structure pool and the fact that the length of each I_i varies from ℓ to 2ℓ (see section 6.1).
(3) For all $i = 1, \ldots, k$, we find the occurrences ending in I_i. We take prefix P_i and suffix S_{j-1} for some suitable $j < i$, such that P_i and S_{j-1} *overlap* for at least $p - 4\ell > 6\ell$ positions (see Fig. 4). More specifically:
   (3.1) We compute $n_i = \sum_{r=1}^{i-1} |I_r|$ for all i (where $n_1 = 0$). For each I_i, we find its *mate* interval I_j, such that j is the minimum integer satisfying $n_i - n_j < p - |I_i|$ (and so $n_i - n_j = |I_j \cdots I_{i-1}| > p - 4\ell$).
   (3.2) We list all the occurrences in $P_i I_i$ by deploying the large overlapping between P_i and S_{j-1}. Namely, we find the occurrences ending in I_i

and contained in the text substring covered by P_i and S_{j-1} when $|P_i|, |S_{j-1}| > n_i - n_j$. We do this by executing a procedure CPS which is described in section 6.2.

Algorithm `Find`'s correctness is illustrated in the following two cases.

If the pattern is short ($p \leq 10\ell$), then every occurrence is a substring of a $string(I_i) \in \Delta_S$, and so the pattern is found by using GST_S in step (1). Even though some occurrences might be listed more than once, each of them can appear no more than 14 times. Thus step (1) can be performed in $O(p + occ)$ time by using an uninitialized boolean vector for discarding all the multiple occurrences. (We do not discuss this case in the rest of the paper.)

If the pattern is long ($p > 10\ell$), then we use the other data structures (i.e., GST_C, GST_R, and GST_L). We scan $\mathcal{L}$ (see [20, 13, 14]) and deploy the bounded length of $\mathcal{L}$'s intervals. Let us now examine an occurrence *ending* in I_i. Its starting position is to the left of I_j because $n_i - n_j < p - |I_i|$ (see Fig. 4). We decompose this occurrence in two overlapping substrings $\hat{P}$ and $\hat{S}$, such that $\hat{P}$ is both a pattern prefix and a suffix of $I_1 \cdots I_{i-1}$, and $\hat{S}$ is both a pattern suffix and a prefix of $I_j \cdots I_k$. By definition, $\hat{P}$ must be a suffix of P_i and $\hat{S}$ must be a prefix of S_{j-1}. Therefore, the occurrence lies entirely within the text substring covered by P_i and S_{j-1} and hence is subsequently listed in step (3.2).

Remark 4.1. Step (2) requires $O(p + n)$ time when the KMP automaton [24] is used. We could obtain better time, i.e., $O(p + \frac{n}{\ell} \log p)$, if the border tree data structure [20] is used. In this paper, we drop the $O(\log p)$ factor by charging $O(1)$ time per character and $\mathcal{L}$'s interval. That is, we achieve $O(p + \ell + \frac{n}{\ell}) = O(p + \frac{n}{\ell})$ time because $p > 10\ell$ (see section 6.1). In this case, the solution is optimal.

Remark 4.2. We can implement step (3.2) in $O(h + p)$ time by simply checking all the candidate positions, where h is the total number of occurrences retrieved. We can obtain an exponential speed-up, i.e., $O(h + \log p)$ time (see [20]), by means of the border tree data structure using the pair (P_i, S_{i-1}). In this paper, we show how to use the pair (P_i, S_{j-1}) in order to drop the $O(\log p)$ factor and yield $\Theta(h+1)$ optimal time (see section 6.2).

In the rest of this paper, we show how to implement steps (2) and (3) *optimally* in $O(p + \frac{n}{\ell})$ and $O(p + \frac{n}{\ell} + occ)$ time in order to achieve our goal of answering `Find`(P) in optimal $O(p + occ)$ time. Indeed, by setting $\ell = \sqrt{n}$, we obtain $O(p + \sqrt{n} + occ)$ query time for a long pattern. Since $p > 10\ell = 10\sqrt{n}$, we achieve the following result.

THEOREM 4.3. *Searching for a pattern $P[1, p]$ in the text $T[1, n]$ requires optimal $O(p + occ)$ worst-case time, where occ is the total number of occurrences.*

4.3. Text updating. In this section, we describe the insertion of a string $Y[1, y]$ into the current text, from a high-level point of view. Our aim is to achieve a *sublinear* update time in the worst case (see section 7 for the technical discussion).

ALGORITHM `Insert`(Y).

(1) We scan $\mathcal{L}$ to find the interval I_i in which the string Y must be inserted to produce $X = I'\, Y\, I''$, where $I_i = I'\, I''$.

(2) We replace I_i with the intervals created from X as follows. We split X into $h = \lfloor \frac{|X|}{\ell} \rfloor = O(y/\ell + 1)$ intervals $I'_1, \ldots, I'_h$, such that $|I'_1| = \cdots = |I'_{h-1}| = \ell$ and $\ell \leq |I_h| < 2\ell$. Then we update $\mathcal{L}$ to obtain the new list $\mathcal{L} = I_1, \ldots, I_{i-1}, I'_1, \ldots, I'_h, I_{i+1}, \ldots, I_k$.

(3) We update the pool $\{GST_S, GST_C, GST_L, GST_R\}$ consistently. In particular, for each set $\Delta \in \{\Delta_S, \Delta_C, \Delta_L, \Delta_R\}$ we have to identify the strings to be deleted and the ones to be inserted because I_i is replaced by $I'_1, \ldots, I'_h$. This process

deletes all the strings in Δ overlapping I_i, and inserts (from scratch) the strings in Δ overlapping at least one of the $h = O(y/\ell + 1)$ new intervals $I'_1, \ldots, I'_h$. Since each interval has a length from ℓ to 2ℓ, the number of strings in Δ overlapping a given interval is a *constant* for all $\Delta \in \{\Delta_S, \Delta_C, \Delta_L, \Delta_R\}$: there are no more than 14 for Δ_S, 2 for Δ_C, 6 for Δ_L, and 6 for Δ_R. This implies that the total number of deleted strings is $O(1)$, while the number of inserted ones is $O(y/\ell+1)$, where each string has $O(\ell)$ length. The data structure pool updating is done by `Add` and `Remove` operations (defined in section 4.1).

We wish to point out that, even if $y < \ell$ and $\ell \leq |X| < 2\ell$, we must insert the whole new interval $I'_1 = X$ from scratch by updating the data structure pool in $O(\ell)$ time.

Deleting a string Y by Algorithm `Delete` is similar to Algorithm `Insert`. We scan $\mathcal{L}$ and remove its shortest sublist $\mathcal{L}'$ that denotes a text substring $I'YI''$. We then delete all the strings in Δ overlapping at least one interval of $\mathcal{L}'$, for each $\Delta \in \{\Delta_S, \Delta_C, \Delta_L, \Delta_R\}$. However, since we do not have to delete I' and I'', we insert them into T again as the string $I'I''$ (if not empty) by Algorithm `Insert` (note that $|I'| + |I''| < 4\ell$). The deletion process therefore removes $O(|\mathcal{L}'|) = O(y/\ell + 1)$ strings and inserts $O(1)$ strings of length $O(\ell)$, for each set $\Delta \in \{\Delta_S, \Delta_C, \Delta_R, \Delta_L\}$.

In the rest of the paper, we provide insertion and deletion procedures that require $O(y + \ell + \frac{n}{\ell})$ time in the worst case (see section 7). By setting $\ell = \sqrt{n}$ (as previously done in the `Find` operation) we obtain the bound we claimed.

THEOREM 4.4. *Inserting a string $Y[1, y]$ in (or deleting it from) $T[1, n]$ requires $O(y + \sqrt{n})$ worst-case time.*

5. Details about the data structure pool. The pool is composed of four main data structures $\{GST_S, GST_C, GST_L, GST_R\}$. In section 4.1, we saw that GST_S and GST_C are generalized suffix trees built on the strings in set $\Delta_S = \{string(I_1), \ldots, string(I_k)\}$ and set $\Delta_C = \cup_{i=1}^{k} C_i$, respectively (we remember that C_i is a set of two strings, namely, I_i's prefix and suffix of length ℓ).

The two other data structures GST_R and GST_L are built on the strings in $\Delta_R = \{R_1, \ldots, R_k\}$ and on the reversal of the strings in $\Delta_L = \{L_1, \ldots, L_k\}$, respectively. Since L_i is the suffix of $I_1 \cdots I_i$ having length 6ℓ, and R_i is the prefix of $I_i \cdots I_k$ having length 6ℓ, the depth of GST_L and GST_R is at most 6ℓ and the number of their nodes is $O(n)$. Although any two equal strings in Δ_L (resp., Δ_R) can share the same leaf in GST_L (resp., GST_R), they cannot be each other's prefix because they have the same length 6ℓ. In what follows, we explain only how to augment GST_R, bearing in mind that the same holds for GST_L.

The leaves of GST_R that only store the strings in Δ_R and not their proper suffixes are *marked*. Two dummy marked leaves (whose parent is the root) are always the leftmost and the rightmost ones in GST_R. A doubly linked list, DLL, maintains the marked leaves in left-to-right order, and its size satisfies $|DLL| \leq |\Delta_R| + 2 = O(\frac{n}{\ell})$. Each leaf f in DLL has an initially empty *push-list*(f) and an initially zero *pop-counter*(f) associated with it. Two pointers LM and RM are associated with all the nodes in GST_R. For a node u, if the leftmost and rightmost marked leaves (i.e., in DLL) descending from u exist, then they can be retrieved by $LM(u)$ and $RM(u)$, respectively. (If those leaves do not exist, then $LM(u) = RM(u) = nil$.) Moreover, the arc linking u to a child v is *marked* whenever v is an ancestor of a marked leaf.

We go on to describe the solution to the operations mentioned in section 4.1 (we recall that the query for GST_S is the ordinary suffix tree search [33]). We also use in our solutions the elegant *matching statistics* that have been presented by Chang and

Lawler [8] and Galil and Giancarlo [16]. They show that, for any given pattern $P[1,p]$, it is possible to build *on-line* two vectors, here called $LONGEST$ and $EXLOCUS$, such that $LONGEST[i]$ contains the *length* of the longest prefix of $P[i,p]$ whose *extended locus* is in GST_Δ, and $EXLOCUS[i]$ contains the pointer to this locus, for $1 \le i \le p$. The on-line computation of these vectors requires a total of $O(p)$ time and space, *provided that* GST_Δ is given.

5.1. Solution to `Query-C`. We now show how to find the pattern substring $SubEq(P,X)$ (if it exists) that is equal to X, for each $X \in \Delta_C$. We use vectors $LONGEST$ and $EXLOCUS$ (described above) computed for the whole pattern and GST_C in $O(p)$ time. The leaves storing the strings in Δ_C, all of length ℓ, can be easily distinguished from the leaves storing their proper suffixes because the latter are always shorter than ℓ. Moreover, no strings longer than ℓ are stored in GST_C. Therefore, if $LONGEST[i] = \ell$, we know that leaf $EXLOCUS[i]$ stores some strings from Δ_C that are equal to $P[i, i+\ell-1]$. Vice versa, if $LONGEST[i] < \ell$, then we can discard $EXLOCUS[i]$ because it points to the extended locus of a *proper* substring of some strings in Δ_C.

We proceed as follows to answer `Query-C`: for $i = 1, \ldots, p$, if $LONGEST[i] = \ell$, then we assign position i to leaf $EXLOCUS[i]$ (unless an integer $j < i$ has already been assigned). Then, for each leaf f having an assigned integer, say i, we retrieve all the strings X whose locus is f (we have $X \in \Delta_C$) and set $SubEq(P,X) := P[i, i+\ell-1]$. At the end, $SubEq(P,X)$ is set equal to the empty string for all the remaining strings $X \in \Delta_C$ that are not involved in the computation above.

LEMMA 5.1. `Query-C(P)` *can be correctly answered in* $O(p + \frac{n}{\ell})$ *time.*

Proof. Let X be a string stored in a leaf, say, f. We know that $X \in \Delta_C$ if and only if $|X| = \ell$. If an integer i is assigned to f, then, by definition of $EXLOCUS$ and $LONGEST$, we have $LONGEST[i] = \ell$. Therefore, $P[i, i+\ell-1]$ can be correctly taken as $SubEq(P,X)$. If an integer is not assigned to f, then either $X \notin \Delta_C$ (because $|X| < \ell$) or $X \in \Delta_C$ and it does not occur in P. The computation of both vectors requires $O(p)$ time [8], and there are $O(\frac{n}{\ell})$ strings in the leaves having an assigned integer (we recall that $|DLL| = O(\frac{n}{\ell})$). □

`Query-C` is particularly useful for performing constant-time comparisons between an interval I_i and a pattern substring, say $P[j, j+|I_i|-1]$. If they are not identical, the computation returns their *longest common prefix* (LCP) in $O(1)$ time. Indeed, let C_i be I_i's cover, given by its prefix I' and its suffix I'' of length ℓ. If either $SubEq(P,I')$ or $SubEq(P,I'')$ is empty, then $I_i \neq P[j, j+|I_i|-1]$. Otherwise, $I_i = P[j, j+|I_i|-1]$ if and only if $SubEq(P,I') = P[j, j+\ell-1]$ and $SubEq(P,I'') = P[j+|I_i|-\ell, j+|I_i|-1]$. These two checks can be performed in constant time through two LCA queries on the suffix tree ST_P (see section 2). If we fail (i.e., $I_i \neq P[j, j+|I_i|-1]$), then we obtain the LCP's length in $O(1)$ time. From now on we refer to that computation as *LCP query*.

5.2. Solution to `Query-R` and `Query-L`. We now show how to find the pattern's *longest suffix* $SufPref(P,X)$ that is a *prefix* of X, for each $X \in \Delta_R$. We take the string $P[p-6\ell+1, p]$ and compute $LONGEST[i]$ and $EXLOCUS[i]$ for all i, such that $p-6\ell+1 \le i \le p$, by Chang and Lawler's algorithm [8] performed on GST_R. (Since $|X| = 6\ell$ for each $X \in \Delta_R$, we only deal with pattern suffixes not longer than 6ℓ.) We then select the set EXN of *distinct* nodes from $EXLOCUS[p-6\ell+1, p]$, such that they are extended loci of *at least one suffix* of $P[p-6\ell+1, p]$. This constraint can be easily verified by checking whether or not $LONGEST[i] = p-i+1$. After that, we associate the position $pos(u) = \min\{i \in [p-6\ell+1, p] : u = EXLOCUS[i]$ and $LONGEST[i] =$

$p-i+1\}$ with each node $u \in EXN$. That is, $pos(u)$ corresponds to the longest suffix of $P[p-6\ell+1, p]$ whose extended locus is u. Finally, the nodes $u \in EXN$ are sorted in *increasing order* according to their $pos(u)$ values. The whole computation is performed in $O(\ell)$ time by scanning $EXLOCUS[p-6\ell+1, p]$ from left to right.

LEMMA 5.2. *Let f be the leaf of GST_R storing a string $X \in \Delta_R$. We have SufPref$(P, X) = P[i, p]$ if and only if $i = pos(u)$ where u is the first of f's ancestors encountered in a left-to-right scan of the sorted set EXN.*

Proof. If $P[i, p]$ is the longest suffix giving SufPref(P, X), then $P[i, p]$'s extended locus is an ancestor of f, say v, which belongs to EXN. Since v is the deepest of f's ancestors having this property, v must be the first node in EXN that is an ancestor of f. We therefore have that $v = u$ and $pos(u) = i$. Conversely, since $i = pos(u)$ and u is an ancestor of f, we infer that $P[i, p]$ is definitely a prefix of X (by the suffix tree's properties). Since u is the first (i.e., deepest) of f's ancestors in EXN, no suffix $P[j, p]$, with $j < i$, can be a prefix of X. Otherwise, $v = EXLOCUS[j]$ would be an ancestor of f with $pos(v) < i$, and so v should occur before u in the sorted EXN, which contradicts our hypothesis. In conclusion, $P[i, p] =$ SufPref(P, X). □

As far as answering `Query-R` is concerned, we only have to apply Lemma 5.2 to the leaves in GST_R that belong to DLL (they store the strings in Δ_R). Since GST_R contains $O(n)$ nodes and has $O(\ell)$ depth, we cannot compute the deepest ancestor in EXN of each leaf in DLL by traversing the tree upwards because it would require $O(|DLL| \cdot \ell)$ time. Instead, we *simulate* this traversal in optimal $O(|DLL|) = O(\frac{n}{\ell})$ time as follows.

We read the *sorted* set EXN from left to right. For each node $u \in EXN$, such that $LM(u), RM(u) \in DLL$, we update the *push-list* in $LM(u)$ and the *pop-counter* in $RM(u)$, as follows: we append $pos(u)$ to the beginning of the *push-list* in $LM(u)$ and increment the *pop-counter* in $RM(u)$. It is worth noting that for any two nodes in EXN, one of which is the other's ancestor, the deepest one is examined first (due to EXN's ordering).

Next, we scan DLL from left to right by means of a stack S. When processing a leaf $f \in DLL$, we push all the elements contained in *push-list*(f) into S (unless *push-list*(f) is empty). We then set SufPref$(P, X) = P[i, p]$ for all $X \in \Delta_R$ stored in f, where i is the integer currently at the top of S (unless S is empty). We execute a pop operation on S for *pop-counter*(f) times, and then go on scanning DLL. At the end, *push-list*(f) and *pop-counter*(f) are reset.

`Query-L` is symmetrical to `Query-R` because PrefSuf$(P, X) =$ SufPref(P^R, X^R). Therefore, we can manage this case by computing $LONGEST$ and $EXLOCUS$ for string $P[1, 6\ell]$'s reversal.

LEMMA 5.3. `Query-L(`P`)` *and* `Query-R(`P`)` *can be correctly answered in $O(\ell + \frac{n}{\ell})$ time.*

Proof. The time complexity readily derives from the preceding considerations. As a matter of fact, we spend $O(\ell)$ time to compute vectors $EXLOCUS$ and $LONGEST$ for $P[p-6\ell+1, p]$, and $O(\frac{n}{\ell})$ time to scan list DLL and vector EXN. We prove the correctness by means of Lemma 5.2 and EXN's ordering. The strategy adopted for setting *push-list*(l) and *pop-counter*(l) (where $l = LM(u)$ or $l = RM(u)$ for $u \in EXN$) allows our simulation to yield the following implicit parenthesis representation of EXN's nodes and DLL's leaves.

We start out with a standard parenthesis representation for trees (e.g., the Euler tour) and apply it to GST_Δ, where $\Delta \in \{\Delta_R, \Delta_L\}$. We cancel all parentheses not corresponding to nodes in EXN or leaves in DLL. The resulting list is still well

balanced. The *nearest pair* of parentheses *enclosing* the pair associated with a leaf $f \in DLL$ corresponds to its deepest ancestor belonging to EXN. We cannot apply this approach directly because its cost would be proportional to GST_Δ's size, i.e., $O(n)$. Consequently, we only deal with the leaves $l \in DLL$ and obtain a construction in $O(\frac{n}{\ell})$ time. Let us assume that $u, v \in EXN$ exist such that $LM(u) = LM(v) = f$, for some $f \in DLL$ (the case of $RM(u) = RM(v) = f$ is simpler). Therefore, u and v must be each other's ancestor, say u ancestor of v. Due to EXN's ordering, v is visited before u in EXN and thus $pos(v)$ is put into the *push-list*(f) before $pos(u)$. Therefore, the *pos*-values associated with each *push-list* maintain the correct nesting of the corresponding parentheses in the Euler tour of GST_Δ. This means that, for every $f \in DLL$, the *pos*-values in *push-list*(f) are in decreasing order (i.e., by their increasing suffix length). Moreover, the counter *pop-counter*(f) is used for balancing the left parentheses with their matching (right) ones.

Let us now assume that $X \in \Delta_R$ is stored in leaf $f \in DLL$. If *push-list*(f) is not empty, the *last* integer i in *push-list*(f) corresponds to the nearest pair of parentheses enclosing f. As a result, this pair of parentheses corresponds, in turn, to the longest suffix $P[i, p]$ that is a prefix of X, as stated by Lemma 5.2. Indeed, integer i is the *pos*-value associated with the first (i.e., deepest) of f's ancestors encountered in EXN. We have that integer i is at the top of S when processing f (because it is the last item in *push-list*(f)). If *push-list*(f) is empty, then none of f's ancestors are in EXN. In brief, since the parentheses are balanced, we always correctly find the integer i associated with the nearest pair of parentheses enclosing the ones for f at the top of S (i.e., the deepest of f's ancestors in EXN). □

5.3. Solution to `Add` and `Remove`. Given a set $\Delta \in \{\Delta_S, \Delta_C, \Delta_L, \Delta_R\}$ and a string $Z[1, z]$, with $z = O(\ell)$, we recall that the operation `Add`(Z, Δ) (resp., `Remove`(Z, Δ)) adds string Z to (resp., removes it from) Δ and updates GST_Δ.

If $\Delta = \Delta_S$ (with $z \leq 14\ell$) or $\Delta = \Delta_C$ (with $z = \ell$), we only have to use the algorithms in [2], which require $O(\ell)$ time. We now discuss only `Add`(Z, Δ_R) without any loss in generality (with $z = 6\ell$). Let Z be a string of length 6ℓ to be added to Δ_R. We insert all of Z's suffixes into GST_R in $O(\ell)$ time by using the algorithm in [2]. Each inserted suffix $Z[i, 6\ell]$ is associated with its locus, namely leaf f_i, for $i = 1, \ldots, 6\ell$. If f_i already exists, then we only have to append $Z[i, 6\ell]$ to the list of (equal) strings associated with f_i. Otherwise, we create f_i and possibly its parent $p(f_i)$. We then update DLL, LM, RM, and the marks in GST_R as follows.

The first suffix (i.e., Z itself) is handled differently from the others. Let Π be the leaf-to-root path from f_1 in GST_R, immediately after the insertion of $Z[1, 6\ell] = Z$. We insert f_1 into the doubly-linked list DLL (because $Z \in \Delta_R$) by finding f_1's predecessor in DLL. We do this by applying a brute-force algorithm that exploits the fact that GST_R cannot be deeper than 6ℓ. Indeed, we go upwards in Π until we reach the deepest node u having an outgoing marked arc to the left of Π. In this case, we follow u's rightmost outgoing marked arc that is to the left of Π and thus we reach a child v. We then insert f_1 into DLL right after $RM(v)$ and update the bi-directional links properly. The insertion of $Z[1, 6\ell]$ into DLL leads to rearranging the pointers LM and RM of some nodes in Π and setting the marks in all of Π's arcs. Finally, we create and initialize *push-list*(f_1) and *pop-counter*(f_1). The overall cost is proportional to GST_R's depth, i.e., $O(\ell)$.

Any other proper suffix $Z[i, 6\ell]$, with $i > 1$, is handled in a simpler way and takes $O(1)$ time each. The pointers LM and RM and the marks associated with the arcs in GST_R do not need to be updated because $Z[i, 6\ell]$ is not inserted into DLL. If

leaf f_i is created, then we set $LM(f_i) = RM(f_i) = nil$. If $p(f_i)$ is also created (by splitting a GST_R's arc), then it can only have two children, namely f_i and another node, say v. The pointers LM and RM in $p(f_i)$ are set equal to v's because $f_i \notin DLL$. If $LM(v), RM(v) \neq nil$, then the arc from $p(f_i)$ to v is marked. (In this case, the arc connecting $p(f_i)$ to its parent has already been marked.) We can now state the following result.

LEMMA 5.4. *Given* $\Delta \in \{\Delta_S, \Delta_C, \Delta_R, \Delta_L\}$ *and a string* Z *of length* $O(\ell)$, `Add(`Z, Δ`)` *and* `Remove(`Z, Δ`)` *can be executed in* $O(\ell)$ *time.*

6. `Find` implementation. We only discuss steps (2) and (3) because step (1) was already described in section 4.2.

6.1. Step (2) of the `Find` algorithm. We remember that we want to compute the pattern's longest prefix P_i that is a suffix of $I_1 \cdots I_{i-1}$ for every $i = 1, \ldots, k$, with $k = \Theta(\frac{n}{\ell})$. (Since finding S_i is symmetrical, we do not discuss it here.) We present a solution requiring $O(p + \ell + \frac{n}{\ell})$ time.

Preprocessing. We execute `Query-L(`P`)` in $O(\ell + \frac{n}{\ell})$ time (Lemma 5.3) in order to compute the pattern's longest prefix *PrefSuf*(P, L_i) that is a suffix of L_i, for all $i = 1, \ldots, k-1$ (see section 4.1 for the definition of L_i). Since $|L_i| = 6\ell$, we have $|\textit{PrefSuf}(P, L_i)| \leq 6\ell$. We also execute `Query-C(`$P$`)` in $O(\ell + \frac{n}{\ell})$ time (Lemma 5.1). We then build the pattern's suffix tree ST_P and preprocess it for handling LCA queries [22, 31] in order to obtain constant-time LCP queries between an interval I_i and a pattern substring (see section 5.1). Finally, we compute the periods of all pattern prefixes in $O(p)$ time by using the algorithm described in [24] (see section 2).

Assuming that P_i has already been found correctly (by induction on $i = 1, \ldots, k-1$, where P_1 is the empty string), we find P_{i+1} in the form of the pattern's longest prefix that is a suffix of $P_i I_i$. We know that P_{i+1} cannot be longer than $P_i I_i$ because otherwise we would have determined a longer P_i. We begin by introducing a simple solution called Algorithm *Naive*.

We check to see if $P_i I_i$ is a pattern prefix by means of an LCP query on I_i and $P[|P_i| + 1, |P_i| + |I_i|]$. If the answer is that the latter two strings are equal, then we set $P_{i+1} := P_i I_i$. Otherwise, we use the *Shift* function in order to examine P_i's borders according to their decreasing lengths (see section 2): we set $P'_i := P_i$ and repeat $P'_i := \textit{Shift}(P'_i)$ until either $P'_i I_i$ is a pattern prefix and $|P'_i| + |I_i| > 6\ell$ or we find $|P'_i| + |I_i| \leq 6\ell$. In the former case, we perform the check in constant time by first making an LCP query that compares I_i to $P[|P'_i| + 1, |P'_i| + |I_i|]$ and then by setting $P_{i+1} := P'_i I_i$ (whenever the LCP query is successful); in the latter case, we set $P_{i+1} := \textit{PrefSuf}(P, L_i)$.

As far as Algorithm *Naive*'s correctness is concerned, we let X denote string concatenation $P'_i I_i$ and assume that X is *not* a pattern prefix. We maintain the invariant that P_{i+1} is a suffix of X by considering two cases. (1) If $|X| > 6\ell$, then P_{i+1}'s prefix (having length $|P_{i+1}| - |I_i| \geq 0$, if it exists) must be a border of P'_i. Thus we can safely take the *longest* border *Shift*(P'_i) to maintain the invariant and continue the computation. (2) If $|X| \leq 6\ell$, then we can correctly set $P_{i+1} := \textit{PrefSuf}(P, L_i)$ because $|P_{i+1}| \leq 6\ell$ (by the invariant). Algorithm *Naive*'s complexity is $O(p + n)$ time because *Shift* is executed for $O(n)$ times in the worst case.

Algorithm *Naive* is inefficient when $|X| > 6\ell$ (and so $|P'_i| \geq 4\ell$) because we can call function *Shift* many times. We let $m = |P'_i| - |\textit{Shift}(P'_i)|$ be the number of characters that are skipped in a *Shift* call. It is worth noting that $m \geq 2\ell$ can occur a total of $O(\frac{n}{m}) = O(\frac{n}{\ell})$ times and so we can pay for its computational cost (this figure appears in the time complexity that we want to obtain in this step). Therefore,

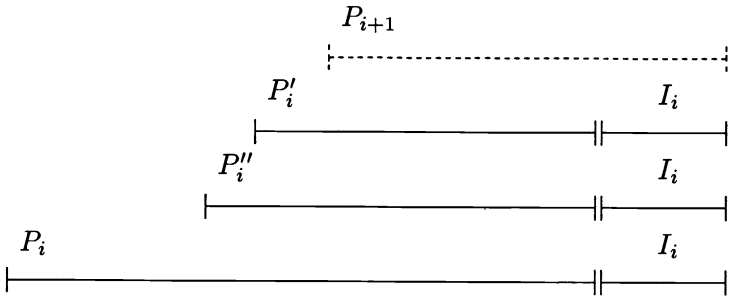


FIG. 5. *The invariant on pattern prefixes P''_i and P'_i.*

$m < 2\ell$ turns out to be Algorithm *Naive*'s most expensive case to deal with. Further on in this paper we prove that, if $|X| > 6\ell$ and $m < 2\ell$, then the period of P'_i is shorter than 2ℓ (i.e., the intervals' maximum length). Hence, we obtain a faster algorithm by exploiting these properties.

We let π denote *the* period of P'_i, such that $|\pi| < 2\ell$, and let P'_i be in the form of $\pi^r\pi'$ for some integer $r \geq 1$. Every border of P'_i has π as *a* period. By examining the borders longer than, or equal to, 4ℓ (by means of some *Shift* calls), we obtain the stronger property that π is actually *the* period of these borders.

LEMMA 6.1. *Every border of P'_i longer than, or equal to, 4ℓ is in the form of $\pi^j\pi'$ for some $j \leq r$ (where $P'_i = \pi^r\pi'$ and $|\pi| < 2\ell$).*

Proof. We take a border $P[1, s]$ of P'_i, with $s \geq 4\ell$, and let $P[1, z]$ be the shortest prefix of P'_i that is in the form $\pi^j\pi'$ (for some $j \leq r$) and contains $P[1, s]$ as prefix (i.e., $z \geq s$ and $z - |\pi| < s$). Our aim is to prove that $z = s$. Let us assume that $z > s$ by way of contradiction. Since both $P[1, z]$ and $P[1, s]$ are borders of P'_i, we deduce that $P[1, s]$ is a suffix of $P[1, z]$. Therefore, $P[1, z - s]$ is a period of $P[1, z]$. Since $z - s < |\pi|$, we have $|\pi| + (z - s) < 2|\pi| < 4\ell \leq z$ and so we can apply Lemma 2.1 to periods π and $P[1, z - s]$ in order to state that $\hat{\pi} = P[1, gcd(|\pi|, z - s)]$ is also a period of $P[1, z]$, where $|\hat{\pi}| < |\pi|$. However, $|\hat{\pi}|$ divides $|\pi|$ and so $\hat{\pi}$ is *shorter* than π, and this contradicts the fact that π is the period of P'_i. Consequently, we must have $s = z$ and the lemma follows because π is thus the period of $P[1, s]$. □

Lemma 6.1 implies that the borders of P'_i longer than, or equal to, 4ℓ are shifted over P'_i by a *multiple* of $|\pi|$ positions. This proves Corollary 6.2, which provides us with a necessary condition for characterizing the candidates for being P_{i+1} (we only deal with the ones longer than 6ℓ).

COROLLARY 6.2. *Let us take a suffix of $X = P'_iI_i$ longer than 6ℓ that is also a pattern prefix, say $P[1, s + |I_i|]$ with $s \geq 4\ell$. We have: $P[1, s]$ is a border of P'_i and its period is π; therefore, $P[1, s + |I_i|]$ is shifted over X by a* multiple *of $|\pi|$ positions.*

By using Lemma 6.1 and Corollary 6.2, and since $\ell \leq |I_i| < 2\ell$, we are able to simulate Algorithm *Naive*'s ith step more efficiently. We maintain two prefixes P''_i, P'_i of P_i under the following invariant (see Fig. 5):

- P''_i, P'_i are P_i's borders, such that $|P''_i| \geq |P'_i|$ and $|P'_i| + |I_i| \geq |P_{i+1}|$ (i.e., P_{i+1} is a suffix of $X = P'_iI_i$).

The invariant initially holds for $P''_1 = P'_1 = P_1 = \epsilon$, where ϵ denotes the empty string. We illustrate the ith inductive step in detail below and use the macro "`Output(`Y`)`" for the following sequence of instructions: "if $|Y| > 6\ell$ then $P_{i+1} := Y$

else $P_{i+1} := \mathit{PrefSuf}(P, L_i)$; $P''_{i+1} := P'_{i+1} := P_{i+1}$; repeat the case analysis for $i := i + 1$."

STEP(i): CASE ANALYSIS.

(1) We let $X = P'_i I_i$. If X is a pattern prefix, then we execute `Output(`X`)`.
(2) If X is *not* a pattern prefix, then we set $P''_i := P'_i$ and $P'_i := \mathit{Shift}(P'_i)$ (and so $|P''_i| > |P'_i|$). We let X denote the new string concatenation $P'_i I_i$. Three cases follow from this:
  (2.1) Case $|X| \leq 6\ell$. Since we have $|P_{i+1}| \leq 6\ell$ by the invariant, we execute `Output(`ϵ`)`.
  (2.2) Case $|X| > 6\ell$ and $|P''_i| - |P'_i| \geq 2\ell$. Since we have a shift of at least 2ℓ positions (i.e., more than the intervals' maximum length), we can repeat the case analysis (1)–(2).
  (2.3) Case $|X| > 6\ell$ and $|P''_i| - |P'_i| < 2\ell$. We let π be the period of P'_i (we show that $|\pi| < 2\ell$ in Lemma 6.3's proof). We write X in the form of $\pi^q\alpha$ and P in the form of $\pi^t\beta$, where π is not a prefix of α and β and so q and t are maximum.
    (2.3.1) If X has period π (i.e., α is a prefix of π) and $q < t$, then we execute `Output(`X`)`.
    (2.3.2) If X has period π and $q \geq t$, then we check to see if α is a prefix of β. If so, then we execute `Output(`$\pi^t\alpha$`)`; otherwise, we execute `Output(`$\pi^{t-1}\alpha$`)`.
    (2.3.3) If X does not have period π and $q < t$, then we execute `Output(`ϵ`)` (we show that $|P_{i+1}| \leq 6\ell$ in Lemma 6.3's proof).
    (2.3.4) If X does not have period π and $q \geq t$, then we check to see if α is a prefix of β. If so, then we execute `Output(`$\pi^t\alpha$`)`; otherwise, we execute `Output(`ϵ`)` (we show that $|P_{i+1}| \leq 6\ell$ in Lemma 6.3's proof).

LEMMA 6.3. *The case analysis in Step(i) correctly finds P_{i+1}.*

Proof. We prove that the preceding case analysis in Step(i) correctly simulates Algorithm *Naive*'s ith step. We assume that we have already found P_i (the basis trivially holds). By the invariant, P_{i+1} is a suffix of $P'_i I_i$. In Case 1, we check to see if $P_{i+1} = P'_i I_i$. If the check fails, then we go to case (2) and examine P'_i and its largest border $\mathit{Shift}(P'_i)$ (as in Algorithm *Naive*). Cases (2.1) and (2.2) closely follow Algorithm *Naive* when we test condition $|P'_i| + |I_i| \leq 6\ell$ and execute one loop step (for $|P''_i| - |P'_i| \geq 2\ell$), respectively, and so we do not go into their details.

Vice versa, we deal with case (2.3) in detail because it occurs when less than 2ℓ characters are skipped by calling $\mathit{Shift}(P'_i)$ (we recall that this was Algorithm *Naive*'s most expensive case). This readily implies that the period of P'_i is shorter than 2ℓ, and hence the period of $\mathit{Shift}(P'_i)$ is also shorter than 2ℓ. Since we previously set $P''_i := P'_i$ and $P'_i := \mathit{Shift}(P'_i)$, we let π be the period of the new P'_i, with $|\pi| < 2\ell$, and X be the new string concatenation $P'_i I_i$. We also have $X = \pi^q\alpha$ and $P = \pi^t\beta$, such that q and t are maximum (i.e., π^{q+1} is not a prefix of X and π^{t+1} is not a prefix of P). It is worth noting that $q, t \geq 2$ because P'_i is a prefix of both X and P and $|P'_i| \geq 4\ell$.

The rationale in case (2.3) is that either P_{i+1} is in the form of $\pi^j\alpha$ for an integer j (when it is longer than 6ℓ) or $P_{i+1} = \mathit{PrefSuf}(P, L_i)$ (when it is not longer than 6ℓ). We therefore show that determining P_{i+1} amounts to executing `Output(`$\hat{P}$`)` for a string $\hat{P}$ defined as follows: let j be the *largest* integer, such that $\pi^j\alpha$ is both a pattern *prefix* and one of X's *suffixes*, with $0 \leq j \leq q$. If such a j exists, then we

take $\hat{P} = \pi^j\alpha$; otherwise, we set $\hat{P} = \epsilon$. We first prove that P_{i+1} is the pattern prefix returned by `Output`$(\hat{P})$, and then we show how to compute $\hat{P}$. Two cases follow.

• If $|P_{i+1}| > 6\ell$, then P_{i+1} must be shifted over X by a multiple of $|\pi|$ positions (Corollary 6.2). Since $\hat{P} = \pi^j\alpha$ is also shifted over X by a multiple of $|\pi|$ positions and is the longest pattern prefix having this property, we deduce that $P_{i+1} = \hat{P}$ (which is the string returned by `Output`$(\hat{P})$ because it is longer than 6ℓ).

• If $|P_{i+1}| \leq 6\ell$, then $\hat{P}$ is a (maybe empty) border of P_{i+1}, and so $|\hat{P}| \leq |P_{i+1}| \leq 6\ell$. Consequently, `Output`$(\hat{P})$ returns $\mathit{PrefSuf}(P, L_i)$, which is P_{i+1} by Lemma 5.3.

The crucial point in the rest of this proof is to show that cases (2.3.1)–(2.3.4) actually find $\hat{P}$. We distinguish two main cases: X has period π (cases (2.3.1)–(2.3.2)) or X does not have period π (cases (2.3.3)–(2.3.4)).

Let us examine X having period π. We deduce that α is a (maybe empty) prefix of π. In case (2.3.1), the whole string X is a pattern prefix and so $\hat{P} = X$ (and we have $j = q$). In case (2.3.2), when α is β's prefix, $\pi^t\alpha$ is a pattern prefix. We choose $\hat{P} = \pi^t\alpha$ (and so $j = t$) because π^{t+1} cannot be a pattern prefix. When α is not β's prefix, we know that α is π's prefix (since X is periodic) and thus $\pi^{t-1}\alpha$ is a prefix of π^t (and so it is a pattern prefix). We choose $\hat{P} = \pi^{t-1}\alpha$ (and so $j = t-1$) because $\pi^t\alpha$ is not a pattern prefix.

Let us now examine X not having period π. We deduce that α and π cannot be each other's prefix. In case (2.3.3), P_{i+1} is not longer than 6ℓ. The proof is obtained by contradiction. If P_{i+1} were longer than 6ℓ, then we could apply Corollary 6.2 to it and find out that P_{i+1} is shifted over X by a multiple of $|\pi|$ positions and is in the form of $\pi^h\alpha$ for some $h \leq q < t$. Since both $P_{i+1} = \pi^h\alpha$ and π^{h+1} would be pattern prefixes (because $h+1 \leq t$), we would have the contradiction that either α is a prefix of π or vice versa. Therefore, P_{i+1} has length 6ℓ at most and cannot be in the form of $\pi^h\alpha$ for any h. In our case analysis, we choose $\hat{P} = \epsilon$; consequently, `Output`$(\hat{P})$ returns $\mathit{PrefSuf}(P, L_i)$, which is P_{i+1}. In case (2.3.4), when α is β's prefix, $\pi^t\alpha$ is a pattern prefix and X's suffix (because $q \geq t$). Since π^{t+1} cannot be a pattern prefix, we choose $\hat{P} = \pi^t\alpha$ (and so $j = t$) and `Output`$(\hat{P})$ returns P_{i+1}. When α is not β's prefix, we can prove $|P_{i+1}| \leq 6\ell$ by following the proof in case (2.3.3). By contradiction, if P_{i+1} were longer than 6ℓ, then we could apply Corollary 6.2 to it and find out that P_{i+1} is shifted over X by a multiple of $|\pi|$ positions and is in the form of $\pi^h\alpha$ for some integer h. We cannot have $h = t$ because $\pi^t\alpha$ would be a pattern prefix and α would be β's prefix. We cannot have $h > t$ because π^{t+1} cannot be a pattern prefix. Finally, we cannot have $h < t$ because both π^{h+1} and $\pi^h\alpha$ would be pattern prefixes and this would imply that either α is a prefix of π or vice versa. As a result, P_{i+1} has length 6ℓ at most and cannot be in the form of $\pi^h\alpha$ for any h. In our case analysis, we choose $\hat{P} = \epsilon$; consequently, `Output`$(\hat{P})$ returns $\mathit{PrefSuf}(P, L_i)$, which is P_{i+1}.

Summarizing our results, we can say that `Output`$(\hat{P})$ always computes P_{i+1} in the case analysis. Since the invariant is maintained for Step$(i+1)$, the correctness proof follows. □

The efficient implementation of Step(i) requires some further comments. In case (1), we can verify that $X = P'_iI_i$ is a pattern prefix in constant time by means of an LCP query on I_i and $P[|P'_i|+1, |P'_i|+|I_i|]$ (see section 5). In cases (2.3.1)–(2.3.4), we do the following. We know that P'_i is in the form of $\pi^r\pi'$ after pattern preprocessing and $r \geq 2$ because $|\pi| < 2\ell$ and $|P'_i| \geq 4\ell$. This allows us to find out if X has period π in constant time: X has period π if and only if I_i occurs in P at position $|\pi'|+1$ (i.e., I_i extends the periodicity of P'_i and this can be checked by an LCP

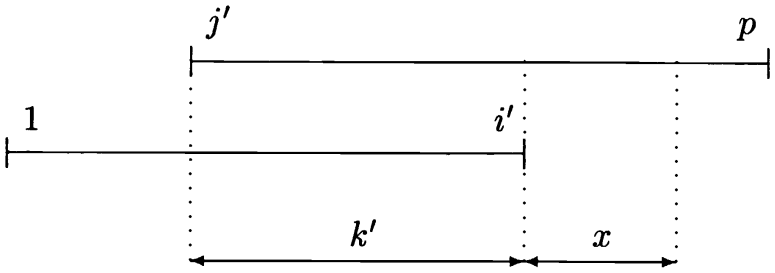


FIG. 6. *The constrained prefix-suffix problem.*

query). We also need to compare q to t in constant time. By an LCP query, we can compute integer q, such that X is in the form of $\pi^q\alpha$. We compute t by performing an LCP query on P and $P[|\pi|+1, p]$.

LEMMA 6.4. *For every $i = 1, \ldots, k$, a total of $O(p + \ell + \frac{n}{\ell})$ time is required for finding the pattern's longest prefix P_i that is a suffix of $I_1 \cdots I_{i-1}$, and the pattern's longest suffix S_i that is a prefix of $I_{i+1} \cdots I_k$.*

Proof. The correctness follows by applying Lemma 6.3 to all the inductive steps. As far as the time complexity is concerned, the preprocessing phase consists of answering `Query-C(`P`)` in $O(p+\frac{n}{\ell})$ time (see Lemma 5.1). Moreover, we execute `Query-L(`P`)` and `Query-R(`P`)` in $O(\ell + \frac{n}{\ell})$ time (see Lemma 5.3). We take $O(p)$ time to build and preprocess ST_P for LCA queries [22, 31]. At a generic step (i), all cases cost $O(1)$ time. Only case (2.2) makes us repeat the case analysis; all other cases give an answer and terminate the computation for the ith step. Therefore, either we perform a shift of at least 2ℓ text positions (case (2.2)) or we find P_{i+1} and increment i in $O(1)$ time (all the other cases). Since $|T| = n$, we perform $O(\frac{n}{\ell})$ shifts at most and increment i for no more than $k = O(\frac{n}{\ell})$ times to visit all of $\mathcal{L}$'s intervals. Therefore, our case analysis takes $O(\frac{n}{\ell})$ time. □

Remark 6.5. It is possible to modify the case analysis of Step(i) to solve the prefix-substring problem [20] in $O(\log p)$ time without using the border tree. The details are left to the reader.

6.2. Step (3) of the `Find` algorithm. As previously mentioned, we want to determine all the occurrences ending at a position inside a given interval I_i. In step (3) (where $p > 10\ell$; see section 4.2), we therefore examined the occurrences in the text substring covered by P_i and S_{j-1}. This computation can be formalized as follows. Given a pattern prefix $P[1, i']$ and a pattern suffix $P[j', p]$, where $1 \le j' \le i' \le p$, the prefix-suffix problem defined in [20] requires us to find all the occurrences in the string concatenation $P[1, i']P[j', p]$. We introduce here a variant of this problem (see Fig. 6) called *constrained prefix-suffix problem* and defined for:

1. two positions i', j', where $1 \le j' \le i' \le p$;
2. an integer $k' > p - 4\ell$, such that $P[i' - k' + 1, i'] = P[j', j' + k' - 1]$ (we have $k' > 6\ell$ because $p > 10\ell$);
3. an interval length x, with $\ell \le x < 2\ell$.

We provide an algorithm $CPS(i', j', k', x)$ to answer the following query: determine all the occurrences in the string $P[1, i']P[j' + k', j' + k' + x - 1]$ obtained by concatenating $P[1, i']$ and one of $P[j', p]$'s substrings of length x. Substep (3.2) in Algorithm `Find` is performed by calling $CPS(i', j', k', x)$ with parameters $i' = |P_i|, j' = p - |S_{j-1}| + 1, k' = n_i - n_j$, and $x = |I_i|$.

We now introduce the following notation regarding $CPS(i', j', k', x)$. We write $P[1, i']$ in the form of $\pi_1^r \pi_1'$ and $P[j', p]$ in the form of $\pi_2^q \pi_2'$ (where π_1 is *the* period of $P[1, i']$, π_2 is *the* period of $P[j', p]$, and π_1', π_2' are prefixes of π_1, π_2, respectively). We indicate the pattern period by π. We go on to prove some properties that are crucial for achieving optimality; they apply to a *long* string $Z = P[1, i']P[j' + k', p]$, with $|Z| > p$.

LEMMA 6.6. *If P occurs in $Z = P[1, i']P[j' + k', p]$ starting at a position in $P[2, i' - k']$ and ending at a position in $P[j' + k', j' + k' + x - 2]$, then $|\pi_1| = |\pi_2| < x < 2\ell$.*

Proof. We let $P' = P[1, i']$ and $S' = P[j', p]$ and assume that there is at least one such occurrence. Since this occurrence starts at $P[2, i' - k']$ and ends in $P[j' + k', j' + k' + x - 2]$, we can consider it a string concatenation $P[1, s]P[s + 1, p]$ (where $s \leq i'$ and $s + x > p$). We deduce that $P'' = P[1, s]$ is a proper suffix of P', and $S'' = P[s - k' + 1, p]$ is a proper prefix of S' (see Fig. 7). Moreover, by letting $K = P[j', j' + k' - 1] = P[i' - k' + 1, i']$ (see point 2), we have $K = P[s - k' + 1, s]$.

We first prove $|\pi_1| < x < 2\ell$ and $|\pi_2| < 4\ell$. By using point 3 and inequality $s + x > p$, we have $i' - s \leq p - s < x < 2\ell$. Since P'' is a border of P', we deduce that $P[1, i' - s]$ is *a period* of P'. By definition, π_1 is the shortest period of P' and so $|\pi_1| \leq i' - s < x < 2\ell$. Furthermore, K is a prefix of both S' and S''. Consequently, the same argument used for π_1 gives us $|\pi_2| < 4\ell$ because $|\pi_2| \leq |S'| - |S''| = s - j' + 1 - k' \leq p - k' < 4\ell$ (by point 2).

We then prove $|\pi_1| = |\pi_2|$. By contradiction, let us assume that $|\pi_1| < |\pi_2|$ (the other case, $|\pi_1| > |\pi_2|$, is similar). K is a suffix of P' and a prefix of S', where $|K| > 6\ell$ by point 2. Let $\hat{\pi}_1$ be K's prefix having length $|\pi_1|$ (i.e., $\hat{\pi}_1 = P[i' - k' + 1, i' - k' + |\pi_1|]$ is a cyclic shift of π_1's characters). According to Fact 2.2, $\hat{\pi}_1$ and π_2 are both K's periods. Moreover, $|\hat{\pi}_1| + |\pi_2| < 6\ell < k' = |K|$. Therefore, by applying Lemma 2.1 to periods $\hat{\pi}_1$ and π_2, we conclude that K has a period of length $G = gcd(|\hat{\pi}_1|, |\pi_2|) = gcd(|\pi_1|, |\pi_2|) < |\pi_2|$. Since G divides $|\pi_2|$, we can infer that S' has a period that is *shorter* than *the* period π_2. This contradicts our hypothesis. □

LEMMA 6.7. *There is an occurrence in $Z = P[1, i']P[j' + k', p]$ starting at $P[2, i' - k']$ and ending in $P[j' + k', j' + k' + x - 2]$ if and only if $|\pi| = |\pi_1| = |\pi_2| < x < 2\ell$ and $i' + x > p + |\pi|$, where π is* the *pattern period.*

Proof. ($\Rightarrow$) Let us assume that such an occurrence exists. By Lemma 6.6, we have $|\pi_1| = |\pi_2| < x < 2\ell$. We show $|\pi| = |\pi_1| = |\pi_2|$ by proving that $|\pi| = |\pi_1|$. Since π_2 is a period of $P[j', p]$ and $|\pi_1| = |\pi_2|$, then π_2 is a cyclic shift of π_1 (i.e., $\pi_2 = P[i' - k' + 1, i' - k' + |\pi_1|]$). Since $P[i' - k' + 1, i'] = P[j', j' + k' - 1]$ with $k' > 6\ell$, we can conclude that π_1 is a period of the whole string Z (π_2 extends π_1's periodicity). Moreover, since $P[1, i']$ is a pattern prefix, we have $|\pi_1| \leq |\pi|$. We cannot have $|\pi_1| < |\pi|$ because P occurs in Z and π_1 is a period of Z, and this would imply that π_1 is a shorter period for P. Therefore, $|\pi| = |\pi_1|$, and $i' + x > p + |\pi|$ because the occurrence in Z starts after the first position and ends before the last one (by our hypothesis).

($\Leftarrow$) Let us assume that $|\pi| = |\pi_1| = |\pi_2| < x < 2\ell$ and $i' + x > p + |\pi|$. Since $Z = P[1, i']P[j' + k', p]$ and $K = P[i' - k' + 1, i'] = P[j', j' + k' - 1]$, with $k' > 6\ell$, we can infer that Z is periodic and its period has length $|\pi| < x < 2\ell$. Therefore, there must be some occurrences that are neither a prefix nor a suffix of $P[1, i']P[j' + k', j' + k' + x - 1]$ because $i' + x > p + |\pi|$. In other words, they start at $P[2, i' - k']$ and end in $P[j' + k', j' + k' + x - 2]$. □

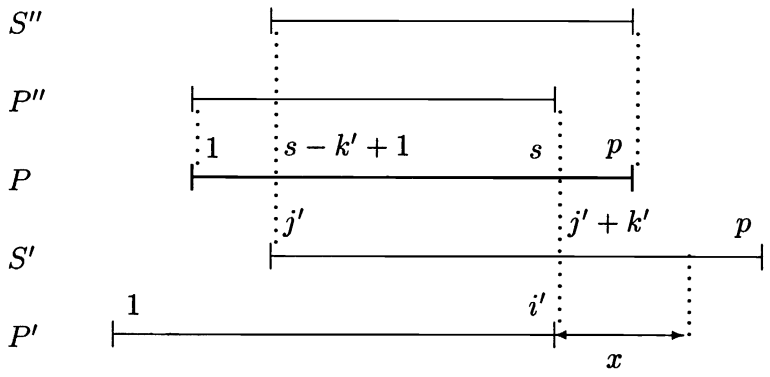


FIG. 7. *A decomposition of a pattern occurrence into its prefix P'' and its suffix S''. Note that P'' (resp., S'') is a border of P' (resp., S').*

COROLLARY 6.8. *If some occurrences in $P[1,i']P[j'+k',p]$ start at $P[2,i'-k']$ and end in $P[j'+k',j'+k'+x-2]$ then they correspond exactly to positions $1+t|\pi|$ in Z, for $t=1,2,\ldots$, such that $t|\pi|+p<i'+x$.*

Proof. From Lemma 6.7, internal occurrences in Z can exist if Z's and P's periods are equal. Therefore, $Z[1+t|\pi|,p+t|\pi|]=P$, for each $t=1,2,\ldots$. Each of these occurrences overlaps both $P[1,i']$ and $P[j'+k',j'+k'+x-2]$ so $t|\pi|+p<i'+x$ makes sure that the occurrences end in $P[j'+k',j'+k'+x-2]$. □

Lemma 6.7 and Corollary 6.8 employ the fact that there is a *large overlapping* $k'>6\ell$ between $P[1,i']$ and $P[j',p]$. We now show how to use them.

Preprocessing. We build and preprocess the suffix tree ST_P in $O(p)$ time in order to answer constant-time LCA queries [22, 28, 31]. Consequently, any two pattern substrings can be compared in constant time by an LCP query (see section 5.1). We also assume that we have previously computed all the pattern prefixes' and suffixes' periods in $O(p)$ time [24]. We let $Pref(P,i')$ (resp., $Suf(P,j')$) denote the length of $P[1,i']$'s (resp., $P[j',p]$'s) period.

ALGORITHM CPS.

Input: Integers i',j',k',x, such that $1\le j'\le i'\le p$, with $p>10\ell$; $k'\ge p-4\ell$ satisfies $P[i'-k'+1,i']=P[j',j'+k'-1]$, and $\ell\le x<2\ell$.

Output: All occurrences in $P[1,i']P[j'+k',j'+k'+x-1]$.

(1) If $i'+x<p$, then we stop because this means that there cannot be any occurrences at all. If $i'+x\ge p$, then we need to find out if P is a prefix or a suffix of $P[1,i']P[j'+k',j'+k'+x-1]$ in constant time. We do it by checking to see if either $P[i'+1,p]$ is a prefix of $P[j'+k',j'+k'+x-1]$ or $P[1,j'+k'-1]$ is a suffix of $P[1,i']$; we use LCP queries for this purpose.

(2) We list the occurrences that are *fully within* $P[1,i']P[j'+k',j'+k'+x-1]$, i.e., that start at $P[2,i'-k']$ and end in $P[j'+k',j'+k'+x-2]$. Specifically, we verify that $|\pi|=Pref(P,i')=Suf(P,j')<x<2\ell$ and $i'+x>p+|\pi|$ in constant time (Lemma 6.7). If so, the occurrences fully within $P[1,i']P[j'+k',j'+k'+x-1]$ are the ones identified by Corollary 6.8 (otherwise, they do not exist).

LEMMA 6.9. *After preprocessing P in $O(p)$ time, each execution of Algorithm CPS correctly finds the h occurrences ending in interval I_i in optimal $\Theta(h+1)$ time.*

Proof. The correctness readily derives from the preceding observations. As far as the time analysis of CPS is concerned, preprocessing takes $O(p)$ time (as previously mentioned). Step (1) requires constant time through LCP queries; step (2) takes $\Theta(h+1)$ time. □

6.3. Analysis of `Find`: Proof of Theorem 4.3. We now prove Theorem 4.3 regarding Algorithm `Find`'s complexity by using the results described in sections 6.1 and 6.2, in which we have two possible cases. If the pattern is short ($p \leq 10\ell$), then GST_S's traversal in step (1) requires $O(p+occ)$ time. If the pattern is long ($p > 10\ell$), then we prove that searching P by means of steps (2) and (3) requires $O(p+\ell+\frac{n}{\ell}+occ)$ time. In step (2), `Query-C(`P`)` requires $O(p+\frac{n}{\ell})$ time (by Lemma 5.1); `Query-R(`P`)` and `Query-L(`P`)` can be answered in $O(\ell+\frac{n}{\ell})$ time (by Lemma 5.3). The case analysis in section 6.1 takes a total of $O(\frac{n}{\ell})$ time plus $O(p)$ preprocessing time (Lemma 6.4). In step (3), we scan $\mathcal{L}$ in $O(\frac{n}{\ell})$ time and list all the occurrences in $O(p+\frac{n}{\ell}+occ)$ time by making $O(\frac{n}{\ell})$ calls to CPS after $O(p)$ preprocessing time (Lemma 6.9). Consequently, searching in steps (2)–(3) takes a total of $O(p+\ell+\frac{n}{\ell}+occ) = O(p+\frac{n}{\ell}+occ)$ time because $p > 10\ell$. By setting $\ell = \sqrt{n}$ (see section 4.2), we prove Theorem 4.3 and the following corollary.

COROLLARY 6.10. *Let Σ be an unbounded ordered alphabet. Searching for a pattern $P[1,p]$ in the text $T[1,n]$ takes $O(p \log|\Sigma| + occ)$ worst-case time, where occ is the total number of occurrences.*

Proof. We store the children list associated with each suffix tree node into a balanced search tree of $O(g)$ size, where $g \leq |\Sigma|$ is the list's size. This implementation introduces a logarithmic slowdown in the data structure pool's traversal and suffix tree construction. The total space required is still $\sum_g O(g) = O(n)$. □

7. `Insert` and `Delete` implementation. Given the interval list $\mathcal{L}$, we want to maintain it consistently under insertion and deletion of new strings. We only deal with insertions here because the same criteria hold for deletions. We refer back to Algorithm `Insert` in section 4.3 and analyze its computational cost. We assume that a string $Y[1,y]$ must be inserted into an interval I_i to produce $X = I'YI''$. In step (1), we scan $\mathcal{L}$ and find the interval I_i in which Y has to be inserted in $O(\frac{n}{\ell})$ time. In step (2), we update $\mathcal{L}$ in $O(y+\ell)$ time by replacing I_i with the intervals $I'_1, \ldots, I'_h$ created from X, where $h = O(y/\ell+1)$. In step (3), $\mathcal{L}$'s change affects sets $\Delta_S, \Delta_C, \Delta_L$, and Δ_R and their generalized suffix trees in the data structure pool (section 4.1). We point out that $O(1)$ strings of length $O(\ell)$ are removed and $O(y/\ell+1)$ strings of length $O(\ell)$ are added to those sets by `Remove` and `Add` operations (see section 5.3). The sets are then updated in a total of $O(y+\ell)$ time (by Lemma 5.4). The cost of `Insert`'s steps amounts to $O(y+\ell+\frac{n}{\ell})$ time.

We wish to add some remarks on insertions and deletions to complete our analysis. Text T can shrink or grow arbitrarily and violate the condition that $\ell = \sqrt{n}$ (we assume that $\sqrt{n}$ is a power of 2 without any loss in generality). Let T_i be the text T after i updates (i.e., string insertions or deletions) and $n_i = |T_i|$, where T_0 is the initial text. We recall that list $\mathcal{L}$ represents T_i's partition into intervals, whose length goes from ℓ to 2ℓ. When $n_i = 4n_0$, parameter ℓ should be *doubled* and list $\mathcal{L}$ recomputed because the intervals are now from 2ℓ to 4ℓ long (so we should set $T_0 := T_i$, $n_0 := n_i$, and $\ell := 2\ell$). The same happens when $n_i = \frac{n_0}{4}$ (i.e., ℓ should be halved). At this point, the reconstruction cost for the data structure pool from scratch could be charged on previous updates' cost but, in this way, the bounds stated in Theorem 4.4 would be *amortized*. Instead, we propose a method based on the "global rebuilding" technique described in [29] in order to obtain worst-case bounds.

Our basic idea consists of maintaining two "partial copies" of the pool and updating them "incrementally." One copy is tuned by parameter 2ℓ and the other by parameter $\frac{\ell}{2}$. For brevity's sake, we only describe how our idea works on $\mathcal{L}$ and GST_S for the copy tuned by parameter 2ℓ.

Given $\mathcal{L} = \{I_1, \ldots, I_k\}$, let us assume that $k/2$ is an integer without any loss in generality, and define $I'_j = I_{2j-1}I_{2j}$ for $j \in [1, k/2]$. It is worth noting that $I'_1 \cdots I'_{k/2}$ represents T_i's partition into intervals of length 2ℓ. We let $string'(I'_j)$ be the prefix of $I'_j \cdots I'_{k/2}$ of length $6 \cdot 2\ell = 12\ell$ (we recall that $string(I_j)$ is the prefix of $I_j \cdots I_k$ of length 6ℓ). Given text T_i, we maintain the following data structures according to a variable $j \in [0, k/2]$:

- a list $\mathcal{L}' = \{I'_1, \ldots, I'_j\}$ (empty for $j = 0$);
- an augmented generalized suffix tree GST'_S on $\Delta'_S = \{string'(I'_1), \ldots, string'(I'_j)\}$ (the set is empty for $j = 0$);
- a garbage area GA containing $O(|I'_{j+1}| + \cdots + |I'_{k/2}|)$ memory cells that we want to dispose of.

At the beginning (for T_0 and $j = 0$), list $\mathcal{L}'$, generalized suffix tree GST'_S, and set Δ'_S are all empty, and GA contains $O(n_0)$ memory cells. When $n_i = 4n_0$, we prove that list $\mathcal{L}'$ is T_i's partition according to the parameter 2ℓ because $j = k/2$. Moreover, at that point, GA is empty and GST'_S stores the whole set $\Delta'_S = \{string'(I'_1), \ldots, string'(I'_{k/2})\}$. As a result, we do not need to reconstruct anything from scratch and we can set $\mathcal{L} := \mathcal{L}'$, $GST_S := GST'_S$, $\Delta_S := \Delta'_S$, and GA to the area containing the "old" pools (i.e., $O(n_i)$ space). We also have to set $T_0 := T_i$, $n_0 := n_i$, $\ell := 2\ell$, $j := 0$, and to reset $\mathcal{L}'$, GST'_S, and Δ'_S to be empty. After that, $|GA| = O(n_0)$ and the invariant is maintained. All this computation requires $O(1)$ worst-case time.

We now show how to maintain the data structures incrementally. Given the insertion or deletion of string $Y[1, y]$, we update $\mathcal{L}$, GST_S, etc., in $O(y + \ell + \frac{n}{\ell})$ time (as shown above). This means adding and removing $O(y/\ell + 1)$ intervals in $\mathcal{L}$. If the intervals in $\mathcal{L}$ have been changed, we must update the corresponding $O(y/\ell + 1)$ intervals in $\mathcal{L}'$. As a consequence, $O(y/\ell + 1)$ strings in Δ'_S and GST'_S are added or removed within the same time bounds. We now introduce the global rebuilding idea. Our updating process also performs some operations that affect the "partial copies" while executing the standard insertion and deletion operations (described above). We append intervals $I'_{j+1}, \ldots, I'_{j+y/\ell+1}$ to $\mathcal{L}'$ by adding $string'(I'_{j+1}), \ldots, string'(I'_{j+y/\ell+1})$ to Δ'_S and by updating GST'_S in $O(y + \ell)$ time by means of Algorithm `Add` (section 5.3). Then we set $j := j + y/\ell + 1$. We also dispose $O(y + \ell)$ memory cells in GA (for some suitable constant factor). This process requires a total of $O(y + \ell + \frac{n}{\ell})$ time that is less than the data structure pool's updating cost.

We now show that everything has been correctly set when $n_i = 4n_0$. Since $|\mathcal{L}| \leq \frac{4n_0}{\ell}$ and some strings (whose total length was $3n_0$) were inserted or deleted (so producing text T_i), at least $2\lceil \frac{3n_0}{\ell} \rceil \geq |\mathcal{L}|$ distinct intervals in $\mathcal{L}$ were used to form list $\mathcal{L}'$ (two intervals in $\mathcal{L}$ form an interval in $\mathcal{L}'$). This allows us to conclude that the final list $\mathcal{L}'$ is the correct partition of T_i according to the parameter 2ℓ because, as soon as $n_i = 4n_0$, we have that $j = k/2$, GA is empty (since we disposed of all the $O(n_0)$ cells), and GST'_S stores $\Delta'_S = \{string'(I'_1), \ldots, string'(I'_{k/2})\}$.

LEMMA 7.1. *Inserting in or deleting a string $Y[1, y]$ from $T[1, n]$ requires $O(y + \ell + \frac{n}{\ell})$ worst-case time.*

Proof. As observed above, we can update the pool and its two "partial copies" and manage the shrinking and growing of the current text incrementally in a total of $O(y + \ell + \frac{n}{\ell})$ time. ∎

By setting $\ell = \sqrt{n}$ (see section 4.3), we obtain Theorem 4.4's proof and the following corollary.

COROLLARY 7.2. *Let Σ be an unbounded ordered alphabet. Inserting in or deleting a string $Y[1, y]$ from $T[1, n]$ requires $O((y + \sqrt{n}) \log |\Sigma|)$ worst-case time.*

8. Conclusions. We studied the problem of searching on-line for the *occ* occurrences of an arbitrary pattern $P[1, p]$ in a text $T[1, n]$ that can change after its preprocessing. We introduced the first dynamic algorithm that achieves optimal query time, i.e., $\Theta(p + occ)$, and sublinear time per update, i.e., $O(\sqrt{n} + y)$ for *every* y, in the worst case. The space required is optimal $\Theta(n)$. This implies that we can achieve the same query time and space usage of suffix trees while improving their update performance.

At present, we do not know if a better $O(t(n)+y)$ update time can be obtained by dynamic text indexing algorithms while maintaining *optimal* query time and *optimal* space usage in the worst case. Our results show that $t(n) = O(\sqrt{n})$. The question is if we can obtain $t(n) = o(\sqrt{n})$ or prove $t(n) = \Omega(\sqrt{n})$ in the worst case. The latter implies that our solution would also be optimal for updates.

Our techniques and data structures may turn out to be useful from a practical point of view. Our search is optimal and we only have to handle some text part of $O(\sqrt{n}+y)$ size for each update operation (e.g., few kilobytes versus several megabytes occupied by the whole text). Moreover, since performance depends on parameter ℓ, we think it might be more convenient in practice to tune ℓ properly by exploiting the fact that p is usually small in real situations. This would allow us to keep optimal query time and faster updating time. Although a long patterns' search would no longer be optimal, we can choose some proper ℓ values to make sure that this actually occurs very rarely.

Acknowledgments. We thank Raffaele Giancarlo and Fabrizio Luccio for their helpful comments and suggestions.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] A. AMIR, M. FARACH, Z. GALIL, R. GIANCARLO, AND K. PARK, *Dynamic dictionary matching*, J. Comput. System Sci., 49 (1994), pp. 208–222.

[3] A. APOSTOLICO, *The myriad virtues of subword trees*, in Combinatorial Algorithms on Words, A. Apostolico and Z. Galil, eds., Springer-Verlag, Berlin, 1985, pp. 85–95.

[4] R. A. BAEZA-YATES, AND G. H. GONNET, *Handbook of Algorithms and Data Structures*, Addison-Wesley, Reading, MA, 1991.

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

[6] A. BLUMER, J. BLUMER, D. HAUSSLER, A. EHRENFEUCHT, M. T. CHEN, AND J. SEIFERAS, *The smallest automaton recognizing the subwords of a text*, Theoret. Comput. Sci., 40 (1985), pp. 31–55.

[7] A. BLUMER, J. BLUMER, D. HAUSSLER, R. MCCONNELL, AND A. EHRENFEUCHT, *Complete inverted files for efficient text retrieval and analysis*, J. Assoc. Comput. Mach., 34 (1987), pp. 578–595.

[8] W. I. CHANG AND E. L. LAWLER, *Sublinear approximate string matching and biological applications*, Algorithmica, 12 (1994), pp. 327–344.

[9] M. T. CHEN AND J. SEIFERAS, *Efficient and elegant subword tree construction*, in Combinatorial Algorithms on Words, A. Apostolico and Z. Galil, eds., Springer-Verlag, Berlin, 1985, pp. 97–107.

[10] M. CROCHEMORE, *Transducers and repetitions*, Theoret. Comput. Sci., 45 (1986), pp. 63–86.

[11] M. Crochemore and W. Rytter, *Parallel construction of minimal suffix and factor automata*, Inform. Process. Lett., 35 (1990), pp. 121–128.
[12] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig, *Sparsification - A technique for speeding up dynamic graph algorithms*, in Proc. IEEE Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 1992, pp. 60–69.
[13] P. Ferragina, *Incremental text editing: A new data structure*, in Proc. European Symposium on Algorithms, Lecture Notes in Computer Science 855, 1994, pp. 495–507; also appeared as *Dynamic text indexing under string updates*, J. Algorithms, 22 (1997), pp. 296–328.
[14] P. Ferragina and R. Grossi, *Fast incremental text editing*, in Proc. ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, PA, 1995, pp. 531–540.
[15] P. Ferragina and R. Grossi, *A fully-dynamic data structure for external substring search*, in Proc. ACM Symposium on Theory of Computing, ACM, New York, 1995, pp. 693–702.
[16] Z. Galil and R. Giancarlo, *Data structures and algorithms for approximate string matching*, J. Complexity, 4 (1988), pp. 33–72.
[17] R. Giancarlo, *A generalization of the suffix tree to square matrices, with applications*, SIAM J. Comput., 24 (1995), pp. 520–562.
[18] R. Giancarlo and R. Grossi, *On the construction of classes of suffix trees for square matrices: Algorithms and applications*, in Proc. International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science 944, 1995, pp. 111–122; also in Inform. and Comput., 130 (1996), pp. 151–182.
[19] G. H. Gonnet, R. A. Baeza-Yates, and T. Snider, *New indices for text: PAT trees and PAT arrays*, in Information Retrieval: Data Structures and Algorithms, W. B. Frakes and R. A. Baeza-Yates, eds., Prentice-Hall, Englewood Cliffs, NJ, 1992, pp. 66–82.
[20] M. Gu, M. Farach, and R. Beigel, *An efficient algorithm for dynamic text indexing*, in Proc. ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, PA, 1994, pp. 697–704.
[21] D. Gusfield, G. M. Landau, and B. Schieber, *An efficient algorithm for all pairs suffix-prefix problem*, Inform. Process. Lett., 41 (1992), pp. 181–185.
[22] H. T. Harel and R. E. Tarjan, *Fast algorithms for finding nearest common ancestors*, SIAM J. Comput., 13 (1984), pp. 338–355.
[23] M. Kempf, R. Bayer, and U. Güntzer, *Time optimal left to right construction of position trees*, Acta Informatica, 24 (1987), pp. 461–474.
[24] D. E. Knuth, J. H. Morris, and V. R. Pratt, *Fast pattern matching in strings*, SIAM J. Comput., 6 (1977), pp. 63–78.
[25] G. M. Landau and U. Vishkin, *Fast parallel and serial approximate string matching*, J. Algorithms, 10 (1989), pp. 157–169.
[26] M. E. Majster and A. Reiser, *Efficient on-line construction and correction of position trees*, SIAM J. Comput., 9 (1980), pp. 785–807.
[27] U. Manber and G. Myers, *Suffix arrays: A new method for on-line string searches*, SIAM J. Comput., 22 (1993), pp. 935–948.
[28] E. M. McCreight, *A space-economical suffix tree construction algorithm*, J. Assoc. Comput. Mach., 23 (1976), pp. 262–272.
[29] M. H. Overmars, *The Design of Dynamic Data Structures*, Lecture Notes in Computer Science 156, Springer-Verlag, Berlin, 1983.
[30] V. Pratt, *Improvements and Applications for the Weiner Repetition Finder*, manuscript, 1975.
[31] B. Schieber and U. Vishkin, *On finding lowest common ancestor: Simplification and parallelization*, SIAM J. Comput., 17 (1988), pp. 1253-1262.
[32] F. N. Teskey, *Principles of Text Processing*, John Wiley, New York, 1983.
[33] P. Weiner, *Linear pattern matching algorithm*, in Proc. IEEE Symposium on Switching and Automata Theory, IEEE, Piscataway, NJ, 1973, pp. 1–11.

Introduction to Special Section on Probabilistic Proof Systems

The study of *probabilistically verifiable proofs* originated in the mid 1980s with the introduction of Interactive Proof Systems (IPs). The primary focus of research in this area in the '80s has been twofold:

- the role of zero-knowledge interactive proofs within cryptographic protocols, and
- characterizing which languages are efficiently interactively provable.

In the 1990s, the focus of research on the topic shifted. Extensions of the interactive proof model, such as Multiprover Interactive Proofs (MIPs) and Probabilistically Checkable Proofs (PCPs), were considered with the intention of expanding our notion of what should be considered efficiently verifiable. In addition, researchers have taken a closer look at the exact resources (and tradeoffs amongst them) needed to verify a proof using various proof systems. This culminated in the important discovery that it is possible to verify NP statements (with a constant error probability) by only examining a constant number of bits of a PCP and using logarithmic amount of randomness.

Perhaps, however, the most dramatic development has been the connection which was found between probabilistically verifiable proofs and proving hardness of approximation for optimization problems. It has been shown that a large variety of optimization versions of NP-hard problems (e.g., the maximum size of a clique in a graph, the minimum number of colors necessary to color a graph, and the maximum number of clauses satisfiable in a CNF formula) are not only NP-hard to solve exactly but also NP-hard to approximate in a very strong sense. The tools to establish hardness of approximation came directly from results on MIPs and PCPs. Indeed, almost every improvement in the efficiency of these proof systems translates directly into showing larger factors within which these optimization problems are hard to approximate.

In 1994–1995 two exciting workshops were held at the Weizmann Institute in Israel on the new developments in probabilistically verifiable proofs and their applications to approximation problems, cryptography, program checking, and complexity theory at large. Over 60 papers were presented in the workshop series, and we are proud to include three of them in this special section.

"On the Power of Finite Automata with Both Nondeterministic and Probabilistic States" by Anne Condon, Lisa Hellerstein, Samuel Pottle, and Avi Wigderson, considers constant round interactive proof systems where the verifier is restricted to use constant space and public coins. An equivalent characterization is finite automata with both nondeterministic and random states (npfa's), which accept their languages with a small probability of error. The paper shows that npfa's restricted to run in polynomial expected time accept only the regular languages in the case of npfa with 1-way input head, and that if L is a nonregular language, then either L or its complement is not accepted by any npfa with a 2-way input head.

"A Parallel Repetition Theorem" by Ran Raz, addresses and resolves the Parallel Repetition Conjecture which has eluded researchers for some time. The broader topic is what happens to the error probability of proof systems when they are composed. It has been known for awhile that sequential composition of proof systems (both single and multiprover interactive proofs) reduces the error exponentially, but this increases the number of rounds. For interactive proof systems, parallel repetition is known to reduce the error exponentially, and the Parallel Repetition Conjecture asserts that the same holds in a *one-round two-prover* proof system. Raz proves a constructive bound on the probability of error which indeed reduces at an exponential rate. The

constant in the exponent is logarithmic in the total number of possible answers of the two provers, which means one can achieve two-prover one-round MIPs for NP statements with arbitrarily small constant error probability. This, in turn, has played a crucial role in further developments in the area and in particular in those reported in the next paper.

"Free Bits, PCPs, and Nonapproximability—Towards Tight Results" by Mihir Bellare, Oded Goldreich, and Madhu Sudan, continues the investigation of PCPs and nonapproximability with emphasis on trying to get closer to tight results. The work consists of three parts. The first part presents several PCP proof systems for NP, based on a new error-correcting code called the Long Code. The second part shows that the connection between PCPs and hardness of approximation is not accidental. In particular, it shows that the transformation of a PCP for NP into hardness results for MaxClique can be reversed. Finally, the third part initiates a systematic investigation of the properties of PCPs as a function of the various parameters: randomness, query complexity, free-bit complexity, amortized free-bit complexity, proof size, etc.

Two more papers submitted for this special section were not ready at this time for publication. They will appear in future issues of the *SIAM Journal on Computing.*

Shafi Goldwasser

SIAM J. COMPUT.
Vol. 27, No. 3, pp. 739–762, June 1998
© 1998 Society for Industrial and Applied Mathematics
010

ON THE POWER OF FINITE AUTOMATA WITH BOTH NONDETERMINISTIC AND PROBABILISTIC STATES*

ANNE CONDON†, LISA HELLERSTEIN‡, SAMUEL POTTLE§, AND AVI WIGDERSON¶

Abstract. We study finite automata with both nondeterministic and random states (npfa's). We restrict our attention to those npfa's that accept their languages with a small probability of error and run in polynomial expected time. Equivalently, we study Arthur–Merlin games where Arthur is limited to polynomial time and constant space.

Dwork and Stockmeyer [*SIAM J. Comput.*, 19 (1990), pp. 1011–1023] asked whether these npfa's accept only the regular languages (this was known if the automaton has only randomness or only nondeterminism). We show that the answer is yes in the case of npfa's with a 1-way input head. We also show that if L is a nonregular language, then either L or $\bar{L}$ is not accepted by any npfa with a 2-way input head.

Toward this end, we define a new measure of the complexity of a language L, called its 1-tiling complexity. For each n, this is the number of tiles needed to cover the 1's in the "characteristic matrix" of L, namely, the binary matrix with a row and column for each string of length $\leq n$, where entry $[x, y] = 1$ if and only if the string $xy \in L$. We show that a language has constant 1-tiling complexity if and only if it is regular, from which the result on 1-way input follows. Our main result regarding the general 2-way input tape follows by contrasting two bounds: an upper bound of polylog(n) on the 1-tiling complexity of every language computed by our model and a lower bound stating that the 1-tiling complexity of a nonregular language or its complement exceeds a function in $2^{\Omega(\sqrt{\log n})}$ infinitely often.

The last lower bound follows by proving that the characteristic matrix of *every* nonregular language has rank n for infinitely many n. This is our main technical result, and its proof extends techniques of Frobenius and Iohvidov developed for Hankel matrices [*Sitzungsber. der Könnigl. Preuss. Akad. der Wiss.*, 1894, pp. 407–431], [*Hankel and Toeplitz Matrices and Forms: Algebraic Theory*, Birkhauser, Boston, 1982].

Key words. nondeterministic probabilistic finite automata, Arthur–Merlin games, interactive proof systems, matrix tiling, Hankel matrices

AMS subject classifications. 68Q05, 68Q10, 68Q75

PII. S0097539794265578

1. Introduction.

The classical subset construction of Rabin and Scott [25] shows that finite state automata with just nondeterministic states (nfa's) accept exactly the regular languages. Results of Rabin [24], Dwork and Stockmeyer [7], and Kaņeps and Freivalds [17] show that the same is true of probabilistic finite state automata which run in polynomial expected time. Here and throughout the paper, we restrict attention to automata which accept languages with error probability that is some constant ϵ less than 1/2.

*Received by the editors March 30, 1994; accepted for publication (in revised form) June 21, 1995. http://www.siam.org/journals/sicomp/27-3/26557.html

†Department of Computer Sciences, University of Wisconsin, 1210 West Dayton Street, Madison, WI 53706 (condon@cs.wisc.edu). The research of this author was supported by NSF grant CCR-9257241 and by a matching grant from AT&T Bell Labs.

‡Department of Computer and Information Sciences, Polytechnic University, 5 Metrotech Center, Brooklyn, NY 11201 (hstein@pucs4.poly.edu). The research of this author was supported in part by NSF grant CCR-9210957.

§Department of Computer Sciences, University of Wisconsin, 1210 West Dayton Sreet, Madison, WI 53706 (pottle@cs.wisc.edu). The research of this author was supported by NSF grant CCR-9257241.

¶Computer Science Department, Hebrew University, Jerusalem, 91904, Israel (avi@cs.huji.ac.il). The research of this author was supported in part by BSF grant 92-00106/1 and a grant from the Wolfson Research Awards.

However, there has been little previous work on finite state automata which have both probabilistic and nondeterministic states. Such automata are equivalent to the Arthur–Merlin games of Babai and Moran [3], restricted to constant space, with an unbounded number of rounds of communication between Arthur and Merlin. In this paper, we refer to them as npfa's. In the computation of an npfa, each transition from a probabilistic state is chosen randomly according to the transition probabilities from that state, whereas from a nondeterministic state, it is chosen so as to maximize the probability that an accepting state is eventually reached. We let 1NPFA and 2NPFA-polytime denote the classes of languages accepted by npfa's which have a 1-way or 2-way input head, respectively, and which run in polynomial expected time. Dwork and Stockmeyer [8] asked whether 2NPFA-polytime is exactly the set of regular languages, which we denote by *Regular*.

In this paper, we prove the following two results on npfa's.

THEOREM 1.1. *1NPFA = Regular.*

THEOREM 1.2. *If L is nonregular, then either L or $\bar{L}$ is not in 2NPFA-polytime.*

Thus, we resolve the question of Dwork and Stockmeyer for npfa's with 1-way head, and in the case of the 2-way head model, we reduce the question to that of deciding whether 2NPFA-polytime is closed under complement. Theorem 1.1 also holds even if the automaton has universal as well as nondeterministic and probabilistic states. Moreover, Theorem 1.2 holds even for Arthur–Merlin games that use $o(\log \log n)$ space.

In proving the two results, we introduce a new measure of the complexity of a language L called its 1-*tiling complexity*. Tiling complexity arguments have been used previously to prove lower bounds for communication complexity (see, e.g., Yao [29]). With each language $L \subseteq \Sigma^*$, we associate an infinite binary matrix M_L, whose rows and columns are labeled by the strings of Σ^*. Entry $M_L[x, y]$ is 1 if the string $xy \in L$ and is 0 otherwise. Denote by $M_L(n)$ the finite submatrix of M_L, indexed by strings of length $\leq n$. Then, the 1-tiling complexity of L (and of the matrix $M_L(n)$) is the minimum size of a set of 1-*tiles* of $M_L(n)$ such that every 1-valued entry of $M_L(n)$ is in at least one 1-tile of the set. Here, a 1-tile is simply a submatrix (whose rows and columns are not necessarily contiguous) in which all entries have value 1.

In section 3, we prove the following theorems, relating language acceptance of npfa's to tiling complexity. The proofs of these theorems build on previous work of Dwork and Stockmeyer [8] and Rabin [24].

THEOREM 3.1. *A language L is in 1NPFA only if the 1-tiling complexity of L is $O(1)$.*

THEOREM 3.4. *A language L is in 2NPFA-polytime only if the 1-tiling complexity of L is bounded by a polynomial in $\log n$.*

What distinguishes our work on tiling is that we are interested in the problem of tiling the matrices $M_L(n)$, which have distinctive structural properties. If L is a unary language, then $M_L(n)$ is a matrix in which all entries along each diagonal from the top right to the bottom left are equal. Such a matrix is known as a *Hankel matrix*. An elegant theory on properties of such Hankel matrices has been developed [15], from which we obtain strong bounds on the rank of $M_L(n)$ if L is unary. In the case that L is not a unary language, the pattern of 0's and 1's in $M_L(n)$ is not as simple as in the unary case, although the matrix still has much structure. Our main technical contribution, presented in section 4, is to prove new lower bounds on the rank of $M_L(n)$ when L is not unary. Our proof uses techniques of Frobenius and Iohvidov developed for Hankel matrices [11], [15].

THEOREM 4.11. *If L is nonregular, then the rank of $M_L(n)$ is at least $n+1$ infinitely often.*

By applying results from communication complexity relating the rank of a matrix to its tiling complexity, we can obtain a lower bound on the 1-tiling complexity of non-regular languages.

THEOREM 4.12. *If L is nonregular, then the 1-tiling complexity of either L or $\bar{L}$ exceeds a function in $2^{\Omega(\sqrt{\log n})}$ infinitely often.*

However, there are nonregular languages, even over a unary alphabet, with 1-tiling complexity $O(\log n)$ (see section 4). Thus the above lower bound on the 1-tiling complexity of L or $\bar{L}$ does not always hold for L itself. A simpler theorem holds for regular languages.

THEOREM 4.2. *The 1-tiling complexity of L is $O(1)$ if and only if L is regular.*

By combining these theorems on the 1-tiling complexity of regular and nonregular languages with the theorems relating 1-tiling complexity to acceptance by npfa's, our two main results (Theorems 1.1 and 1.2) follow as immediate corollaries.

The rest of the paper is organized as follows. In Section 2, we define our model of the npfa and the tiling complexity of a language. We conclude that section with a discussion of related work on probabilistic finite automata and Arthur–Merlin games. In section 3, we present Theorems 3.1 and 3.4, which relate membership of a language L in the classes 1NPFA and 2NPFA-polytime to the 1-tiling complexity of L. A similar theorem is presented for the class 2NPFA, in which the underlying automata are not restricted to run in polynomial expected time. In section 4, we present our bounds on the tiling complexity of both regular and nonregular languages. Theorems 1.1 and 1.2 are immediate corollaries of the main results of sections 3 and 4. Extensions of these results to alternating automata and to Turing machines with small space are presented in section 5. Conclusions and open problems are discussed in section 6.

2. Preliminaries. We first define our npfa model in section 2.1. This model includes as special cases the standard models of nondeterministic and probabilistic finite state automata. In section 2.2 we define our notion of the tiling complexity of a language. Finally, in section 2.3, we discuss previous work on this and related models.

2.1. Computational models and language classes. A *two-way nondeterministic probabilistic finite automaton* (2npfa) consists of a set of states Q, an input alphabet Σ, and a transition function δ, with the following properties. The states Q are partitioned into three subsets: the *nondeterministic states* N, the *probabilistic* (or *random*) states R, and the *halting states* H. H consists of two states: the *accepting state* q_a and the *rejecting state* q_r. There is a distinguished state q_0, called the *initial state*. There are two special symbols $\text{¢}, \$ \notin \Sigma$, which are used to mark the left and right ends of the input string, respectively.

The transition function δ has the form

$$\delta : Q \times (\Sigma \cup \{\text{¢}, \$\}) \times Q \times \{-1, 0, 1\} \to \{0, 1/2, 1\}.$$

For each fixed q in R, the set of random states, and $\sigma \in (\Sigma \cup \{\text{¢}, \$\})$, the sum of $\delta(q, \sigma, q', d)$ over all q' and d equals 1. The meaning of δ in this case is that if the automaton is in state q reading symbol σ, then with probability $\delta(q, \sigma, q', d)$ the automaton enters state q' and moves its input head one symbol in direction d (left if $d = -1$, right if $d = 1$, stationary if $d = 0$). For each fixed q in N, the set of nondeterministic states, and $\sigma \in (\Sigma \cup \{\text{¢}, \$\})$, $\delta(q, \sigma, q', d) \in \{0, 1\}$ for all q'

and d. The meaning of δ in this case is that if the automaton is in state q reading symbol σ, then the automaton nondeterministically chooses some q' and d such that $\delta(q, \sigma, q', d) = 1$, enters state q' and moves its input head one symbol in direction d. Once the automaton enters state q_a (respectively, q_r), the input head moves repeatedly to the right until the right endmarker \$ is read, at which point the automaton halts. In other words, for $q \in \{q_a, q_r\}$, $\delta(q, \sigma, q, 1) = 1$ for all $\sigma \in \Sigma \cup \{\mathfrak{c}\}$, and $\delta(q, \sigma, q', 1) = 0$ for all $\sigma \in \Sigma \cup \{\mathfrak{c}\}$ and $q' \neq q$. On a given input, the automaton is started in the *initial configuration*, that is, in the initial state with the head at the left end of the input. If the automaton halts in state q_a on the input, we say that it *accepts* the input, and if it halts in state q_r, we say that it *rejects* the input.

Fix some input string $w = w_0 w_1 w_2, \ldots, w_n w_{n+1}$, where $w_0 = \mathfrak{c}$ and $w_{n+1} = \$$. A *nondeterministic strategy* (or just *strategy*) on w is a function

$$S_w : N \times \{0, \ldots, n+1\} \to Q \times \{-1, 0, 1\}$$

such that $\delta(q, \sigma, q', d) = 1$ whenever $S_w(q, j) = (q', d)$ and $w_j = \sigma$. The meaning of S_w is that if the automaton is in state $q \in N$ reading w_j, then if $S_w(q, j) = (q', d)$, the automaton enters state q' and moves its input head one symbol in direction d. The strategy indicates which nondeterministic choice should be made in each configuration.

A language $L \subseteq \Sigma^*$ is accepted with *bounded error probability* if for some constant $\epsilon < 1/2$,

1. for all $w \in L$, there exists a strategy S_w on which the automaton accepts with probability $\geq 1 - \epsilon$, and
2. for all $w \notin L$, on every strategy S_w, the automaton accepts with probability $\leq \epsilon$.

Language acceptance could be defined with respect to a more general type of strategy, in which the nondeterministic choice made from the same configuration at different times may be different. It is known (see [4, Theorem 2.6]) that if L is accepted by an npfa with respect to this more general definition, then it is also accepted with respect to the definition above. Hence, our results also hold for such generalized strategies.

A *one-way nondeterministic probabilistic finite automaton* (1npfa) is a 2npfa which can never move its input head to the left; that is, $\delta(q, \sigma, q', -1) = 0$ for all q, q', and σ. Also, a *probabilistic finite automaton* (pfa) and a *nondeterministic finite automaton* (nfa) are special cases of an npfa in which there are no nondeterministic and no probabilistic states, respectively.

We denote by 1NPFA and 2NPFA the classes of languages accepted with bounded error probability by 1npfa's and 2npfa's, respectively. If, on all inputs w and all nondeterministic strategies, the 2npfa halts in polynomial expected time, we say that L is in the class 2NPFA-polytime. The classes 1PFA, 2PFA, and 2PFA-polytime are defined similarly, with pfa replacing npfa. Finally, Regular denotes the class of regular languages.

Our model of the 2npfa is equivalent to an Arthur–Merlin game in which Arthur is a 2pfa, and our classes 2NPFA and 2NPFA-polytime are identical to the classes AM(2pfa) and AM(ptime-2pfa), respectively, of Dwork and Stockmeyer [8].

2.2. The tiling complexity of a language. We adapt the notion of the tiling complexity of a function, used in communication complexity theory, to obtain a new measure of the complexity of a language. Given a finite, two-dimensional matrix M, a *tile* is a submatrix of M in which all entries have the same value. A tile is specified by a pair (R, C) where R is a nonempty set of rows and C is a nonempty set of columns.

The entries in the tile are said to be *covered* by the tile. A tile is a b-tile if all entries of the submatrix are b. A set of b-tiles is a *b-tiling* of M if every b-valued entry of M is covered by at least one tile in the set. If M is a binary matrix, the union of a 0-tiling and a 1-tiling of M is called a *tiling* of M. Let $T(M)$ be the minimum size of a tiling of M. Let $T^1(M)$ be the minimum size of a 1-tiling of M, and let $T^0(M)$ be the minimum size of a 0-tiling of M. Then, $T(M) = T^1(M) + T^0(M)$. Note that in these definitions it is permitted for tiles of the same type to overlap.

We can now define the tiling complexity of a language. Associated with a language L over alphabet Σ is an infinite binary matrix M_L. The rows and columns of M_L are indexed (say, in lexicographic order) by the strings in Σ^*. Entry $M_L[x, y] = 1$ if and only if $xy \in L$. Let L_n be the strings of L of length $\leq n$. Let $M_L(n)$ be the finite submatrix of M_L whose rows and columns are indexed by the strings of length $\leq n$. The 1-*tiling complexity* of a language L is defined to be the function $T^1_L(n) = T^1(M_L(n))$. Similarly, the 0-*tiling complexity* of L is $T^0_L(n) = T^0(M_L(n))$ and the *tiling complexity* of L is $T_L(n) = T(M_L(n))$.

A tiling of a matrix M is *disjoint* if every entry $[x, y]$ of M is covered by exactly one tile. The disjoint tiling complexity of a matrix M, $\tilde{T}(M)$, is the minimum size of a disjoint tiling of M. Also, the disjoint tiling complexity of a language, $\tilde{T}_L(n)$, is $\tilde{T}(M_L(n))$.

Tilings are often used in proving lower bounds in communication complexity [29], [30]. Let $f : X \times Y \to \{0, 1\}$. The function f is represented by a matrix M_f whose rows are indexed by elements of X and whose columns are indexed by elements of Y, such that $M_f[x, y] = f(x, y)$. Let T_f denote $T(M_f)$. Suppose that two cooperating parties, P_1 and P_2, get inputs $x \in X$ and $y \in Y$, respectively, and want to compute $f(x, y)$. They can do so by exchanging information according to some protocol (precise definitions of legal protocols can be found in [13]). If the protocol is deterministic, then the worst case number of bits that need to be exchanged (that is, the deterministic communication complexity) is bounded below by $\log \tilde{T}_f$ [29]. If the protocol is nondeterministic, then the lower bound is $\log T_f$ [1]. Finally, if the object of the nondeterministic protocol is only to verify that $f(x, y) = 1$ (if that is indeed the case), then the lower bound on the number of bits exchanged is $\log T^1_f$.

2.3. Related work. Our work on npfa's builds upon a rich literature on probabilistic finite state automata. Rabin [24] was the first to consider probabilistic automata with bounded error probability. He showed that 1PFA = Regular. However, with a 2-way input head, pfa's can recognize nonregular languages. This was shown by Freivalds [10], who constructed a 2pfa for the language $\{0^n 1^n \mid n \geq 0\}$. Greenberg and Weiss [12] showed that exponential expected time is required by any 2pfa accepting this language. Dwork and Stockmeyer [7] and, independently, Kaņeps and Freivalds [17] showed that, in fact, any 2pfa which recognizes a nonregular language must run in exponential expected time. It follows that 2PFA-polytime = Regular.

Roughly, Rabin's proof shows that any language L accepted by a 1pfa has only finitely many equivalence classes. Here, two strings x, x' are equivalent if and only if for all y, $xy \in L \Leftrightarrow x'y \in L$. The Myhill-Nerode theorem [14] states that a language has a finite number of equivalence classes if and only if it is regular. This, combined with Rabin's result, implies that 1PFA = Regular. Two decades later, this idea was extended to 2pfa's. A strengthened version of the Myhill-Nerode theorem is needed for this extension. Given a language L, we say that two strings x, x' are *pairwise n-inequivalent* if for some y, $xy \in L \Leftrightarrow x'y \notin L$, and furthermore, $|xy|, |x'y| \leq n$. Let $N_L(n)$ (the *nonregularity* of L) be the size of the largest set of pairwise n-inequivalent

strings. Kaŋeps and Freivalds [16] showed that $N_L(n) \geq \lfloor (n+3)/2 \rfloor$ for infinitely many n. (It is interesting to note that, to prove their bound, Kaŋeps and Freivalds first showed that $N_L(n)$ equals the number of states of the minimal deterministic 1-way finite automaton that accepts all words of length $\leq n$ that are in L and rejects all words of length $\leq n$ that are not in L. Following Karp [19], we denote the latter measure by $\phi_L(n)$. Karp [19] previously proved that $\phi_L(n) > n/2 + 1$ for infinitely many n. Combining this with the fact that $N_L(n)$ and $\phi_L(n)$ are equal, it follows immediately that $N_L(n) > n/2 + 1$ for infinitely many n. This is stronger (by 1) for even n than the lower bound of Kaŋeps and Freivalds. We also note that Dwork and Stockmeyer [7] obtained a weaker bound on $N_L(n)$ without using $\phi_L(n)$. Using tools from Markov chain theory, Dwork and Stockmeyer [7] and Kaŋeps and Freivalds [17] showed that if a language is accepted by a 2pfa in polynomial expected time, then the language has "low" nonregularity. In fact, $N_L(n)$ is bounded by some polynomial in $\log n$. This, combined with the result of Kaŋeps and Freivalds, implies that 2PFA-polytime = Regular.

Models of computation with both nondeterministic and probabilistic states have been studied intensively since the work of Papadimitriou [23] on games against nature. Babai and Moran [3] defined Arthur–Merlin games to be Turing machines with both nondeterministic and probabilistic states, which accept their languages with bounded error probability. Their work on polynomial time bounded Arthur–Merlin games laid the framework for the remarkable progress on interactive proof systems and their applications (see, for example, [2] and the references therein). Space bounded Arthur–Merlin games were first considered by Condon and Ladner [6]. Condon [4] showed that AM(log-space), that is, the class of languages accepted by Arthur–Merlin games with logarithmic space, is equal to the class P. However, it is not known whether the class AM(log-space, polytime)—the subclass of AM(log-space) where the verifier is also restricted to run in polynomial time—is equal to P, or whether it is closed under complement. Fortnow and Lund [9] showed that NC is contained in AM(log-space, polytime).

Dwork and Stockmeyer [8] were the first to consider npfa's, which are Arthur–Merlin games restricted to constant space. They described conditions under which a language is not in either of the classes 2NPFA or 2NPFA-polytime. The statements of our Theorems 3.2 and 3.4 generalize and simplify the statements of their theorems, and our proofs build on theirs. In communication complexity theory terms, their proofs roughly show that languages accepted by npfa's have low "fooling set complexity." This measure is defined in a manner similar to the tiling complexity of a language, based on the following definition. Define a 1-fooling set of a binary matrix A to be a set of entries $\{[x_1, y_1], [x_2, y_2], \ldots, [x_m, y_m]\}$ such that $A[x_i, y_j] = 1$ if and only if $i = j$. The size of a 1-fooling set of a binary matrix is always at most the 1-tiling complexity of the matrix, because no two distinct entries in the 1-fooling set, $[x_i, y_i]$ and $[x_j, y_j]$, can be in the same tile. However, the 1-tiling complexity may be significantly larger than the 1-fooling set complexity; in fact, for a random $n \times n$ binary matrix, the expected size of the largest 1-fooling set is $O(\log n)$, whereas the expected number of tiles needed to tile the 1-entries is $\Omega(n/\log n)$ [1].

3. NPFA's and tiling. Three results are presented in this section. For each of the classes 1NPFA, 2NPFA, and 2NPFA-polytime, we describe upper bounds on the tiling complexity of the languages in these classes. The proof for 1NPFA's is a natural generalization of Rabin's proof that 1PFA = Regular [24]. The other two proofs build on previous results of Dwork and Stockmeyer [8] on 2npfa's.

3.1. 1NPFA and tiling.

THEOREM 3.1. *A language L is in 1NPFA only if the 1-tiling complexity of L is $O(1)$.*

Proof. Suppose L is accepted by some 1npfa M with error probability $\epsilon < 1/2$. Let the states of M be $\{1, \ldots, c\}$.

Consider the matrix M_L. For each 1-entry $[x, y]$ of M_L, fix a nondeterministic strategy that causes the string xy to be accepted with probability at least $1 - \epsilon$. With respect to this strategy, define two vectors of dimension c. Let $\mathbf{p}_{xy}$ be the *state probability vector* at the step when the input head moves off the right end of x. That is, the ith entry of the vector is the probability of being in state i at that moment, assuming that the automaton is started at the left end of the input $\mathfrak{c}\, xy\$$ in the initial state. Let $\mathbf{r}_{xy}$ be the column vector whose ith entry is the probability of accepting the string xy, assuming that the automaton is in state i at the moment that the head moves off the right end of x. Then the probability of accepting the string xy is the inner product $\mathbf{p}_{xy} \cdot \mathbf{r}_{xy}$.

Let $\mu = (1/2 - \epsilon)/c$. Partition the space $[0, 1]^c$ into *cells* of size $\mu \times \mu \times \cdots \times \mu$ (the final entry in the cross product should actually be less than μ if 1 is not a multiple of μ). Associate each 1-entry $[x, y]$ with the cell containing the vector $\mathbf{p}_{xy}$; we say that $[x, y]$ belongs to this cell.

With each cell C, associate the *rectangle* R_C defined as

$$\{x| \text{ there exists } y \text{ such that } [x, y] \text{ belongs to } C\} \times \{y| \text{ there exists } x \text{ such that } [x, y] \text{ belongs to } C\}.$$

This is the minimal submatrix that covers all of the entries associated with cell C.

We claim that R_C is a valid 1-tile; that is, R_C covers only 1-entries. To see this, suppose $[x, y] \in R_C$. If $[x, y]$ belongs to C, then it must be a 1-entry. Otherwise, there exist x' and y' such that $[x, y']$ and $[x', y]$ belong to C; that is, $xy', x'y \in L$, and $\mathbf{p}_{xy'}$ and $\mathbf{p}_{x'y}$ are in the same cell.

We claim that xy is accepted with probability at least $1/2$ on some strategy, namely, the strategy that, while reading x, uses the strategy for xy', and while reading y, uses the strategy for $x'y$. To see this, note that

$$\begin{aligned}(\mathbf{p}_{x'y} - \mathbf{p}_{xy'}) \cdot \mathbf{r}_{x'y} &= \sum_{i=1}^{c} [\mathbf{p}_{x'y} - \mathbf{p}_{xy'}]_i [\mathbf{r}_{x'y}]_i \\ &\leq \mu \sum_{i=1}^{c} [\mathbf{r}_{x'y}]_i \\ &\leq \mu c \\ &= 1/2 - \epsilon, \quad \text{by our choice of } \mu.\end{aligned}$$

Hence, the probability that xy is accepted on the strategy described above is

$$\begin{aligned}\mathbf{p}_{xy'} \mathbf{r}_{x'y} &\geq \mathbf{p}_{x'y} \mathbf{r}_{x'y} - (1/2 - \epsilon) \\ &\geq (1 - \epsilon) - (1/2 - \epsilon) \\ &= 1/2 > \epsilon.\end{aligned}$$

Because xy is accepted with probability greater than ϵ on this strategy, it cannot be that $xy \notin L$. Hence, for all $[x, y] \in R_C$, xy must be in L. Therefore R_C is a 1-tile in M_L.

Every 1-entry $[x, y]$ is associated with some cell C and is covered by the 1-tile R_C that is associated with C. Thus, every 1-entry of M_L is covered by some R_C.

Hence, L can be 1-tiled using one tile per cell, which is a total of $\lceil 1/\mu \rceil^c = O(1)$ tiles. □

3.2. 2NPFA and tiling. We next show that if $L \in$ 2NPFA, then $T_L^1(n)$ is bounded by a polynomial.

THEOREM 3.2. *A language L is in 2NPFA only if the 1-tiling complexity of L is bounded by a polynomial in n.*

Proof. Suppose L is accepted by some 2npfa M with error probability $\epsilon < 1/2$. Let c be the number of states of M. As in Theorem 3.1, for each 1-entry $[x, y]$ of $M_L(n)$, fix a nondeterministic strategy that causes M to accept the string xy with probability at least $1 - \epsilon$.

We construct a stationary Markov chain H_{xy} that models the computation of M on xy using this strategy.

This Markov chain has $d = 2c + 4$ states. $2c$ of the states are labeled (q, l), where q is a state of M and $l \in \{0, 1\}$. The other states are labeled Initial, Accept, Reject, and Loop. The state $(q, 0)$ of H_{xy} corresponds to M being in state q while reading the rightmost symbol of $\mathfrak{c}\, x$. The state $(q, 1)$ of H_{xy} corresponds to M being in state q while reading the leftmost symbol of $y\$$. The state Initial corresponds to the initial configuration of M. The states Accept, Reject, and Loop are sink states of H_{xy}.

A single step of the Markov chain H_{xy} corresponds to running M on input xy (using the fixed nondeterministic strategy) from the appropriate configuration for one or more steps until M enters a configuration corresponding to one of the chain states (q, l). If M halts in the accepting (respectively, rejecting) state before entering one of these configurations, H_{xy} enters the Accept (respectively, Reject) state. If M does not halt and never again reads the rightmost symbol of $\mathfrak{c}\, x$ or the leftmost symbol of $y\$$, then H_{xy} enters the Loop state. The transition probabilities are defined accordingly.

Consider the transition matrix of H_{xy}. Collect the rows corresponding to the chain states Initial and $(q, 0)$ (for all q) and call this submatrix P_{xy}. Collect the rows corresponding to the chain states $(q, 1)$ and call this submatrix R_{xy}. Then the transition matrix looks like this:

$$H_{xy} = \begin{array}{r|c|c} \text{Initial} & \multicolumn{2}{c}{\multirow{2}{*}{P_{xy}}} \\ (q,0) & \multicolumn{2}{c}{} \\ \hline (q,1) & \multicolumn{2}{c}{R_{xy}} \\ \hline \begin{array}{r}\text{Accept}\\ \text{Reject}\\ \text{Loop}\end{array} & 0 & I_3 \end{array},$$

where I_3 denotes the identity matrix of size 3. (We shall engage in a slight abuse of notation by using H_{xy} to refer to both the transition matrix and the Markov chain itself.) Note that the entries of P_{xy} depend only on x and the nondeterministic strategy used; these transition probabilities do not depend on y. This assertion appears to be contradicted by the fact that our choice of nondeterministic strategy may depend on y; however, the idea here is that if we replace y with y' while maintaining the same nondeterministic strategy we used for xy, then $P_{xy'}$ will be identical to P_{xy}, because

the transitions involved simulate computation of M on the left part of its input only. Similarly, R_{xy} depends only on y and the strategy, and not on x.

We now show that if $|x| \leq n$ and if p is a nonzero element of P_{xy}, then $p \geq 2^{-cn-1}$. Form a second Markov chain $K(\mathfrak{c}\, x)$ with states of the form (q, l), where q is a state of M and $1 \leq l \leq |\mathfrak{c}\, x| + 1$. The chain state (q, l) with $l \leq |\mathfrak{c}\, x|$ corresponds to M being in state q scanning the lth symbol of $\mathfrak{c}\, x$. Transition probabilities from these states are obtained from the transition probabilities of M in the obvious way. Chain states of the form $(q, |\mathfrak{c}\, x| + 1)$ are sink states of $K(\mathfrak{c}\, x)$ and correspond to the head of M falling off the right end of $\mathfrak{c}\, x$ with M in state q. Now consider a transition probability p in P_{xy}. Suppose that, in the Markov chain H_{xy}, p is the transition probability from $(q, 0)$ to $(q', 1)$. Then $p \in \{0, 1/2, 1\}$, since if H_{xy} makes this transition, it must be simulating a single computation step of M. Suppose p is the transition probability from $(q, 0)$ to $(q', 0)$. If $p > 0$, then there must be some path of nonzero probability in $K(\mathfrak{c}\, x)$ from state $(q, |\mathfrak{c}\, x|)$ to $(q', |\mathfrak{c}\, x|)$ that visits no state $(q'', |\mathfrak{c}\, x|)$, and since $K(\mathfrak{c}\, x)$ has at most cn states that can be on this path, there must be such a path of length at most $cn + 1$. Since $1/2$ is the smallest nonzero transition probability of M, it follows that $p \geq 2^{-cn-1}$. The cases where p is a transition probability from the Initial state are similar.

Similarly, if $|y| \leq n$ and if r is a nonzero element of R_{xy}, then $r \geq 2^{-cn-1}$.

Next we present a lemma that bounds the effect of small changes in the transition probabilities of a Markov chain. This lemma is a slight restatement of a lemma of Greenberg and Weiss [12]. This version is due to Dwork and Stockmeyer [8].

If k is a sink state of a Markov chain R, let $a(k, R)$ denote the probability that R is (eventually) trapped in state k when started in state 1. Let $\beta \geq 1$. Say that two numbers r and r' are *β-close* if either: (i) $r = r' = 0$, or (ii) $r > 0, r' > 0$, and $\beta^{-1} \leq r/r' \leq \beta$. Two Markov chains $R = \{r_{ij}\}_{i,j=1}^{s}$ and $R' = \{r'_{ij}\}_{i,j=1}^{s}$ are *β-close* if r_{ij} and r'_{ij} are β-close for all pairs i, j.

LEMMA 3.3. *Let R and R' be two s-state Markov chains which are β-close, and let k be a sink state of both R and R'. Then $a(k, R)$ and $a(k, R')$ are β^{2s}-close.*

The proof of this lemma is based on the Markov chain tree theorem of Leighton and Rivest [20] and can be found in [8].

Our approach is to partition the 1-entries of $M_L(n)$ into equivalence classes, as in the proof of Theorem 3.1, but this time we will make entries $[x, y]$ and $[x', y']$ equivalent only if the corresponding Markov chains H_{xy} and $H_{x'y'}$ are β-close, where β will be chosen small enough that we can use Lemma 3.3 to show that xy' and $x'y$ are accepted with high probability by combining the strategies for xy and $x'y'$.

If $[x, y]$ is a 1-entry such that $|x| \leq n$ and $|y| \leq n$, then for any nonzero p of P_{xy} (or r of R_{xy}), $p \in [2^{-cn-1}, 1]$, so $\log_2 p \in [-cn - 1, 0]$ (and similarly $\log_2 r \in [-cn - 1, 0]$).

By partitioning each coordinate interval $[-cn - 1, 0]$ into subintervals of length μ, we divide the space $[-cn - 1, 0]^{d^2}$ into at most $\lceil (cn + 1)/\mu \rceil^{d^2}$ cells, each of size at most $\mu \times \mu \times \cdots \mu$.

Partition the 1-entries in $M_L(n)$ into equivalence classes by making xy and $x'y'$ equivalent if H_{xy} and $H_{x'y'}$ have the property that for each state transition, if p and p' are the respective transition probabilities, either $p = p' = 0$, or $\log p$ and $\log p'$ are in the same (size μ) subinterval of $[-cn - 1, 0]$.

Note that the number of equivalence classes is at most $(\lceil (cn + 1)/\mu \rceil + 1)^{d^2}$.

We claim that if μ is chosen small enough, these equivalence classes induce a 1-tiling of $M_L(n)$ of size at most the number of equivalence classes. As in Theorem

3.1, we associate with each equivalence class C the rectangle R_C defined by

$$\{x|\text{there exists } y \text{ such that } [x,y] \in C\} \times \{y|\text{there exists } x \text{ such that } [x,y] \in C\}.$$

We claim that for each $[x,y]$ in R_C, $xy \in L$. That is, all entries in the rectangle are 1, so the rectangle forms a 1-tile. Let $[x,y]$ be in R_C. There must be some y' such that $[x,y'] \in C$ and some x' such that $[x',y] \in C$. Consider the associated Markov chains $H_{xy'}$ and $H_{x'y}$, and in particular, consider the transition submatrices $P_{xy'}$ and $R_{x'y}$. The first is associated with a particular nondeterministic strategy on x, namely, one which assumes the input is xy' and tries to cause xy' to be accepted with high probability. The second is associated with a particular nondeterministic strategy on y, namely, one which assumes the input is $x'y$ and tries to cause $x'y$ to be accepted with high probability. The two matrices $P_{xy'}$ and $R_{x'y}$ taken together correspond to a hybrid strategy on xy: while reading x, use the strategy for xy', and while reading y, use the strategy for $x'y$. We will argue that this hybrid strategy causes xy to be accepted with probability $\geq 1/2$.

We construct a hybrid Markov chain H_{xy} using $P_{xy'}$ and $R_{x'y}$. This chain models the computation of M on xy using the hybrid strategy.

Since the 1-entries $[x,y']$ and $[x',y]$ are in the same equivalence class C, it follows that if p and p' are corresponding transition probabilities in the Markov chains $H_{xy'}$ and $H_{x'y}$, then either $p = p' = 0$ or $|\log p - \log p'| \leq \mu$. Therefore, $H_{xy'}$ and $H_{x'y}$ are 2^μ-close, and it immediately follows that H_{xy} is 2^μ-close to $H_{xy'}$ (and to $H_{x'y}$). Let $a_{xy'}$ be the probability that M accepts input xy' on the strategy for xy', and let a_{xy} be the probability that M accepts input xy using the hybrid strategy. Then $a_{xy'}$ (respectively, a_{xy}) is exactly the probability that the Markov chain $H_{xy'}$ (respectively, H_{xy}) is eventually trapped in the Accept state when started in the Initial state. Now $xy' \in L$ implies $a_{xy'} \geq 1 - \epsilon$. Since H_{xy} and $H_{xy'}$ are 2^μ-close, Lemma 3.3 implies that

$$\frac{a_{xy}}{a_{xy'}} \geq 2^{-2d\mu},$$

which implies

$$a_{xy} \geq (1-\epsilon)2^{-2d\mu}.$$

Since ϵ and d are constants, and since $\epsilon < 1/2$, we can choose μ to be a constant so small that $a_{xy} \geq 1/2$. Therefore xy must be in L.

Since each 1-entry $[x,y]$ is in some equivalence class, the matrix $M_L(n)$ can be 1-tiled using at most $(\lceil (cn+1)/\mu \rceil + 1)^{d^2}$ tiles. Therefore,

$$T_L^1(n) \leq (\lceil (cn+1)/\mu \rceil + 1)^{d^2}.$$

Since c, d, and μ are constants independent of n, this shows that $T_L^1(n)$ is bounded by a polynomial in n. □

3.3. 2NPFA-polytime and tiling. We now show that if $L \in$ 2NPFA-polytime, then $T_L^1(n)$ is bounded by a polylog function.

THEOREM 3.4. *A language L is in 2NPFA-polytime only if the 1-tiling complexity of L is bounded by a polynomial in $\log n$.*

Proof. Suppose L is accepted by some 2npfa M with error probability $\epsilon < 1/2$ in expected time at most $t(n)$. Let c be the number of states of M. For each 1-entry $[x,y]$ of $M_L(n)$, fix a nondeterministic strategy that causes M to accept the string xy with probability at least $1 - \epsilon$.

We construct the Markov chain H_{xy} just as in Theorem 3.2.

Say that a probability p is *small* if $p < t(n)^{-2}$; otherwise, p is *large*. Note that if p is a large transition probability, then $p \in [t(n)^{-2}, 1]$, so $\log_2 p \in [-2\log_2 t(n), 0]$. When dividing the 1-entries of $M_L(n)$ into equivalence classes, make xy and $x'y'$ equivalent if H_{xy} and $H_{x'y'}$ have the property that for each state transition, if p and p' are the respective transition probabilities, either p and p' are both small, or $\log p$ and $\log p'$ are in the same (size μ) subinterval of $[-2\log_2 t(n), 0]$.

This time the number of equivalence classes is at most $(\lceil 2\log_2 t(n)/\mu \rceil + 1)^{d^2}$.

Model the computation of M on inputs $x'y$, xy', and xy by Markov chains $H_{x'y}$, $H_{xy'}$, and H_{xy}, respectively, as before.

If p and p' are corresponding transition probabilities in any two of these Markov chains, then either p and p' are 2^{μ}-close or p and p' are both small. Let $\mathcal{E}_{x'y}$ be the event that, when $H_{x'y}$ is started in state Initial, it is trapped in state Accept or Reject before any transition labeled with a small probability is taken; define $\mathcal{E}_{xy'}$ and $\mathcal{E}_{xy}$ similarly. Since M halts in expected time at most $t(n)$ on the inputs $x'y$, xy', and xy, the probabilities of these events go to 1 as n increases. Therefore, by changing all small probabilities to zero, we do not significantly change the probabilities that $H_{x'y}$, $H_{xy'}$, and H_{xy} enter the Accept state, provided that n is sufficiently large. A formal justification of this argument can be found in Dwork and Stockmeyer [8].

After these changes, we can argue that

$$a_{xy} \geq (1-\epsilon)2^{-2d\mu}$$

and choose μ so that $a_{xy} \geq 1/2$, as before. It then follows that

$$T_L^1(n) \leq (\lceil 2\log_2 t(n)/\mu \rceil + 1)^{d^2} \tag{1}$$

for all sufficiently large n, establishing the result. □

4. Bounds on the tiling complexity of languages. In this section, we obtain several bounds on the tiling complexity of regular and nonregular languages. In section 4.1, we prove several elementary results. First, all regular languages have constant tiling complexity. Second, the 1-tiling complexity of all nonregular languages is at least $\log n - O(1)$ infinitely often. We also present an example of a (unary) nonregular language which has 1-tiling complexity $O(\log n)$. In section 4.2, we use a rank argument to show that for all nonregular languages L, either L or its complement has "high" 1-tiling complexity infinitely often.

4.1. Simple bounds on the tiling complexity of languages. The following lemma is useful in proving some of the theorems in this section. Its proof is implicit in work of Melhorn and Schmidt [21]; we include it for completeness.

LEMMA 4.1. *Any binary matrix A that can be 1-tiled with m tiles has at most 2^m distinct rows.*

Proof. Let A be a binary matrix that can be 1-tiled by m tiles $\{T_1, \ldots, T_m\}$, where $T_j = (R_j, C_j)$. For each row r of A, let $I(r) = \{T_j \mid j \in \{1, \ldots, m\}$ such that $r \in R_j\}$. Suppose r_1 and r_2 are rows such that $I(r_1) = I(r_2)$. We show that in this case, rows r_1 and r_2 are identical. To see this, consider any column c of A. Suppose that entry $[r_1, c]$ has value 1 and is covered by some tile $T_j \in I(r_1)$. Therefore, $c \in C_j$. Since $I(r_1) = I(r_2)$, $T_j \in I(r_2)$ and therefore, $r_2 \in R_j$ and $[r_2, c]$ is covered by tile T_j. Hence, entry $[r_2, c]$ must have value 1, since T_j is a 1-tile. Hence, if $[r_1, c]$ has value 1, so does $[r_2, c]$. Similarly, if $[r_2, c]$ has value 1, then so does entry $[r_1, c]$. Therefore r_1 and r_2 are identical rows. Since there are only 2^m possible values for $I(r)$, A can have at most 2^m distinct rows. □

THEOREM 4.2. *The* 1*-tiling complexity of* L *is* $O(1)$ *if and only if* L *is regular.*

Proof. By the Myhill-Nerode theorem [14, Theorem 3.6], L is regular if and only if M_L has a finite number of distinct rows.

Suppose L is regular. Then by the above fact there exists a constant k such that M_L has at most k distinct rows. Consider any (possibly infinite) set R of identical rows in M_L. Let C_b be the set of columns which have bit b in the rows of R, for $b = 0, 1$. Then the subset specified by (R, C_b) is a b-tile and covers all the b-valued entries in the rows of R. It follows that the 1-valued entries of R can be covered by a single tile, and hence there is a 1-tiling of $M_L(n)$ of size k. (Similarly, there is a 0-tiling of $M_L(n)$ of size k.)

Suppose L is not regular. Since L is not regular, M_L has an infinite number of distinct rows. It follows immediately from Lemma 4.1 that M cannot be tiled with any constant number of tiles. □

The above theorem uses the simple fact that the 1-tiling complexity $T_L^1(n)$ of a language L is a lower bound on the number of distinct rows of $M_L(n)$. In fact, the number of distinct rows of $M_L(n)$, for a language L, is closely related to a measure that has been previously studied by many researchers. Dwork and Stockmeyer called this measure *nonregularity*, and denoted the nonregularity of L by $N_L(n)$ [7]. $N_L(n)$ is the maximum size of a set of n*-dissimilar* strings of L. Two strings, w and w', are considered n-dissimilar if $|w| \leq n$ and $|w'| \leq n$, and there exists a string v such that $|wv| \leq n$, $|w'v| \leq n$, and $wv \in L$ if and only if $w'v \notin L$. It is easy to show that the number of distinct rows of $M_L(n)$ is between $N_L(n)$ and $N_L(2n)$. Previously, Kaŋeps and Freivalds [16] showed that $N_L(n)$ is equal to the number of states of the minimal 1-way deterministic finite state automaton which accepts a language L' for which $L'_n = L_n$, where L_n is the set of strings of L of length $\leq n$.

Shallit [28] introduced a similar measure: the *nondeterministic nonregularity* of L, denoted by $NN_L(n)$, is the minimal number of states of a 1-way nondeterministic finite automaton which accepts a language L' for which $L'_n = L_n$. In fact, it is not hard to show that

$$T_L^1(n) \leq NN_L(2n).$$

To see this, suppose that M is an automaton with $NN_L(2n)$ states, which accepts a language L' for which $L'_{2n} = L_{2n}$. We construct a 1-tiling of $M_L(n)$ with one tile T_q per state q of M, where entry $[x, y]$ is covered by T_q if and only if there is an accepting path of M on xy which enters state q as the head falls off the rightmost symbol of x. It is straightforward to verify that the set of tiles defined in this way is indeed a valid 1-tiling of $M_L(n)$. A similar argument was used by Schmidt [27] to prove lower bounds on the number of states in an unambiguous nfa.

We next turn to simple lower bounds on the 1-tiling complexity of nonregular languages. From Theorem 4.2, it is clear that if L is nonregular, then $T_L^1(n)$ is unbounded. We now use a known lower bound on the nonregularity of nonregular languages to prove a lower bound for $T_L^1(n)$.

THEOREM 4.3. *If* L *is not regular, then* $T_L^1(n) \geq \log_2 n - 1$ *for infinitely many* n*.*

Proof. Kaŋeps and Freivalds [16] proved that if L is not regular, then $N_L(n) \geq \lfloor (n+3)/2 \rfloor$ for infinitely many n. By the definition of $N_L(n)$, the matrix $M_L(n)$ must have at least $N_L(n)$ distinct rows. Therefore, by Lemma 4.1, $T_L^1(n) \geq \log_2 N_L(n)$. The lemma follows immediately. □

We next present an example of a unary nonregular language, with 1-tiling complexity $O(\log n)$. Thus, the lower bound of Theorem 4.3 is optimal to within a constant factor.

THEOREM 4.4. *Let L be the complement of the language $\{a^{2^k-1} \mid k > 0\}$. Then, L has 1-tiling complexity $O(\log n)$.*

Proof. We show that the 1-valued entries of $M_L(n)$ can be covered with $O(\log n)$ 1-tiles. Let $\lg n$ denote $\lfloor \log_2 n \rfloor + 1$, and let $\lg 0 = 0$. Let x and y be binary numbers of length at most $\lg n$. Number the bits of these numbers from right to left, starting with 1, so that, for example, $y = y_{\lg n} \dots y_2 y_1$. For any binary number q, $\lg q$ is the maximum index i such that $q_i = 1$ ($\lg q = 0$ if $q = 0$).

Clearly, if q is equal to $2^k - 1$ for some integer $k > 0$, then for all indices $i, 1 \leq i \leq \lg q$, $q_i = 1$. The next fact follows easily.

FACT. *$x + y = 2^k - 1$ for some integer $k > 0$ if and only if for all j such that $j \leq \max\{\lg x, \lg y\}$, $x_j \neq y_j$.*

Roughly, we construct a 1-tiling of $M_L(n)$, corresponding to the following nondeterministic communication protocol. The party P_1 guesses an index j and sends j and x_j to P_2. Also, P_1 sends P_2 one bit indicating whether or not $j \leq \lg x$. If $j \leq \lg x$, then P_2 checks that $y_j = x_j$. If $j > \lg x$, P_2 checks that $j \leq \lg y$ and that $y_j = x_j$, or equivalently, that $y_j = 0$. In either case, P_2 can conclude that $y_j = x_j$, and so entry $[a^x, a^y]$ of $M_L(n)$ is 1. The number of bits sent from P_1 to P_2 is $\lg \lg n + 2$.

We now describe the 1-tiling corresponding to this protocol. It is the union of two sets of tiles. The first set has one tile $T_{j,b}$ for each j, b such that $\lg n \geq j \geq 0$ and $b \in \{0, 1\}$, where

$$T_{j,b} = \{a^x \mid 0 \leq x \leq n, \lg x \geq j, x_j = b\} \times \{a^y \mid 0 \leq y \leq n, y_j = b\}.$$

The second set of tiles has one tile $S_{j,0}$, for all j such that $\lceil \log n \rceil \geq j \geq 1$.

$$S_{j,0} = \{a^x \mid 0 \leq x \leq n, \lg x < j, x_j = 0\} \times \{a^y \mid 0 \leq y \leq n, \lg y \geq j, y_j = 0\}.$$

To see that all the 1's in the matrix are covered by one of these tiles, note that if entry $[a^x, a^y]$ of the matrix is 1, then by the above fact, there exists an index j such that $j \leq \max\{\lg x, \lg y\}$, and either $x_j = y_j = 1$, or $x_j = y_j = 0$. So, for example, if $\lg x \geq \lg y$, and j is such that $j \leq \lg x$ and $x_j = y_j = 0$, then entry $[a^x, a^y]$ is covered by tile $T_{j,0}$. □

The nondeterministic communication protocol in the above proof is a slight variation of a simple (and previously known) protocol for the complement of the set distinctness problem. In the set distinctness problem, each of the two parties holds a subset of $\{1, \dots, m\}$ and must determine whether the subsets are distinct. In our application, the problem is to determine, for $m = \max\{\lg x, \lg y\}$, whether the subset of $\{1, \dots, m\}$, whose corresponding values in x are 0, is distinct from the subset of $\{1, \dots, m\}$ whose corresponding values in y are 1.

4.2. Lower bounds on the tiling complexity of nonregular languages. In this section we prove that if a language L is nonregular, then the 1-tiling complexity of either L or $\bar{L}$ is "high" infinitely often. To prove this, we first prove lower bounds on the rank of M_L when L is nonregular. We then apply theorems from communication complexity relating rank to tiling complexity.

The proofs of the lower bounds on the rank of M_L are heavily dependent on distinctive structural properties of M_L. Consider first the case where L is a unary language over the alphabet $\Sigma = \{a\}$. In this case, for all i, j where $j > 1$, $a^i a^j = a^{i+1} a^{j-1}$, and therefore $M_L[a^i, a^j] = M_L[a^{i+1}, a^{j-1}]$. It follows that for every n, $M_L(n)$ is such that its auxiliary diagonal (the diagonal from the top right to the bottom left) consists of equal elements, as do all diagonals parallel to that diagonal. An example is shown in Figure 1. Such matrices are classically known as *Hankel matrices* and have been extensively studied [15]. In fact, a direct application of known results on the rank of

	ϵ	a^1	a^2	a^3	a^4	a^5	a^6
ϵ	1	0	0	1	0	0	1
a^1	0	0	1	0	0	1	0
a^2	0	1	0	0	1	0	0
a^3	1	0	0	1	0	0	1
a^4	0	0	1	0	0	1	0
a^5	0	1	0	0	1	0	0
a^6	1	0	0	1	0	0	1

FIG. 1. *The Hankel matrix* $M_L(6)$ *for* $L = \{a^i | i \equiv 0 \bmod 3\}$.

	ϵ	0	1	00	01	10	11	000	001	010	011	100	101	110	111
ϵ	1	1	1	1	0	0	1	1	0	1	0	0	1	0	1
0	1	1	0	1	0	1	0	1	0	0	0	0	0	1	0
1	1	0	1	0	1	0	1	0	1	0	0	0	0	0	1
00	1	1	0	1	0	0	0	1	0	0	0	1	0	0	0
01	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0
10	0	0	1	0	1	0	0	0	1	0	0	0	1	0	0
11	1	**0**	1	**0**	**0**	0	1	**0**	**0**	**0**	**1**	0	0	0	1
000	1	1	0	1	0	0	0	1	0	0	0	0	0	0	0
001	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0
010	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0
011	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0
100	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0
101	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0
110	**0**	**0**	**0**	**0**	**0**	**0**	**1**	0	0	0	1	0	0	0	0
111	1	0	1	0	0	0	1	0	0	0	0	0	0	0	1

FIG. 2. *The matrix* $M(3)$ *for* $L = \{w \in \{0,1\}^* | w \text{ is a palindrome}\}$. *The bold entries in row* 110 *are determined by the bold entries in row* 11. *The bold entries in row* 110 *comprise* $split^{(0)}(11)$ *for* $M(2,3)$.

Hankel matrices shows that if L is nonregular, then $\text{rank}(M_L(n)) \geq n+1$ infinitely often. This was first proved by Iohvidov (see [15, Theorem 11.3]), based on previous work of Frobenius [11].

If L is a nonunary language, then M_L does not have the simple diagonal structure of a Hankel matrix. Nevertheless, M_L still has structural properties that we are able to exploit. In fact, the term Hankel matrix has been extended from its classical meaning to refer to matrices M_L of nonunary languages (see [26]). In what follows, we generalize the results on the rank of classical Hankel matrices and prove that for any nonregular language L, over an arbitrary alphabet, $\text{rank}(M_L(n)) \geq n+1$ infinitely often.

4.2.1. Notation and basic facts. Let L be a language over an arbitrary alphabet, and let $M = M_L$.

Consider a row of M indexed by a string w. This row corresponds to strings that have the prefix w. For any string s, row ws corresponds to strings with the prefix ws. Thus the entries in row ws can be determined by looking at those entries in row w whose columns are indexed by strings beginning with s (see Figure 2). In what follows, we consider this relationship between the rows of M more formally.

Let $M(n,m)$ denote the set of vectors (finite rows) of M which are indexed by strings x of length $\leq n$ and whose columns are indexed by strings of length $\leq m$. Let $\hat{M}(n,m)$ denote the subset of vectors of $M(n,m)$ which are indexed by strings x of

length exactly n. If v' is row x of $M(n, m+i)$, where $i > 0$ and v is row x of $M(n, m)$, then v' is called an *extension* of v.

Suppose $v \in M(n, m)$. Let s be a string over Σ of length $\leq m$ (possibly the empty string, ϵ). Define $\text{split}^{(s)}(v)$ to be the subvector formed from v by selecting exactly those columns whose labels have s as a prefix. Also, relabel the columns of $\text{split}^{(s)}(v)$ by removing the prefix s. Note that $\text{split}^{(\epsilon)}(v) = v$. Note also that if Σ is unary, say $\{\sigma\}$, then $\text{split}^{(\sigma)}(v)$ is v with the first column removed. Let $|v|$ denote the dimension (number of entries) of vector v. If Σ is binary and $\sigma \in \Sigma$, then

$$|\ \text{split}^{(\sigma)}(v)| = (|v| - 1)/2.$$

More generally, if $|\Sigma| = c > 1$ and $\sigma \in \Sigma$, then

$$|v| = \frac{c^{m+1} - 1}{c - 1}, \text{ and}$$
$$|\ \text{split}^{(\sigma)}(v)| = \frac{|v| - 1}{c} = \frac{c^m - 1}{c - 1}.$$

Also, the vector v consists of the first entry (indexed by the empty string, ϵ), plus an "interleaving" of the entries of $\text{split}^{(\sigma)}(v)$, for each $\sigma \in \Sigma$. More precisely, we have the following fact.

FACT 4.1. *Let $j', s, j \in \Sigma^*$, where $j' = sj$. Then, $v[j'] = \text{split}^{(s)}(v)[j]$.*

We generalize the definition of the split function to sets of vectors. If V is a set of vectors in $M(n, m)$, and $|s| \leq m$, let $\text{split}^{(s)}(V) = \{\ \text{split}^{(s)}(v) \mid v \in V\}$. Then we have the following.

FACT 4.2. $\cup_{|s|=i}\text{split}^{(s)}(\hat{\text{M}}(n, m)) = \hat{\text{M}}(n + i, m - i)$. *Thus,*

(a) $\hat{\text{M}}(n + i, m - i) \subseteq \cup_{|s|=i}\text{split}^{(s)}(M(n, m))$, *and*

(b) $\cup_{|s|=i}\text{split}^{(s)}(M(n, m)) = M(n + i, m - i)$.

In what follows, the vectors we consider are assumed to be elements of vector spaces over an arbitrary field $\mathbf{F}$ (e.g., our proofs will hold if $\mathbf{F}$ is taken to be the field of rationals $\mathbf{F}$). All references to rank, span, and linear independence apply to vector spaces over $\mathbf{F}$.

LEMMA 4.5. *Suppose that $b_1, \ldots, b_p \in M(n, m)$ and that*

$$v = \alpha_1 b_1 + \cdots + \alpha_p b_p,$$

where the α_i are in the field $\mathbf{F}$. Suppose that for $1 \leq k \leq p$, b'_k is an extension in $M(n, m + 1)$ of b_k and that v' is an extension of v to the same length as the b'_k.

Suppose also that for some $i, 0 \leq i \leq m + 1$, it is the case that for all s of length i,

$$\text{split}^{(s)}(v') = \alpha_1\text{split}^{(s)}(b'_1) + \cdots + \alpha_p\text{split}^{(s)}(b'_p).$$

Then $v' = \alpha_1 b'_1 + \cdots + \alpha_p b'_p$.

Proof. Clearly, $v'[j] = \alpha_1 b'_1[j] + \cdots + \alpha_p b'_p[j]$, if j is a string of length $\leq m$. Consider a string j' of length $m + 1$. Let $j' = sj$, where $|s| = i$. By Fact 4.1,

$$v'[j'] = \ \text{split}^{(s)}(v')[j].$$

Also,

$$b'_k[j'] = \ \text{split}^{(s)}(b'_k)[j], \quad \text{for } 1 \leq k \leq p.$$

By the hypothesis of the lemma,

$$\text{split}^{(s)}(v')[j] = \alpha_1 \text{ split}^{(s)}(b'_1)[j] + \cdots + \alpha_p \text{ split}^{(s)}(b'_p)[j].$$

Putting the last three equalities together, $v'[j'] = \alpha_1 b'_1[j'] + \cdots + \alpha_p b'_p[j']$, as required. □

Let $\text{rank}(M(n,m))$ be the rank of the set of vectors $M(n,m)$ and let $\text{span}(M(n,m))$ be the vector space generated by the vectors in $M(n,m)$. The next lemma follows immediately from the definitions.

LEMMA 4.6. *If* $v' \in \text{span}(M(n,m)), m > 0$ *and* $v = \text{split}^{(\sigma)}(v')$, *where* $\sigma \in \Sigma$, *then*

$$v \in \text{span}(\text{split}^{(\sigma)}(M(n,m))).$$

4.2.2. A lower bound on the rank of $M(n)$ when L is nonregular. A trivial lower bound on the rank of $M(n)$ is given by the following fact.

FACT 4.3. *L is nonregular if and only if there is an infinite sequence of integers p_r satisfying* $\text{rank}(M(p_r)) \geq r+1$ *for all integers r.*

This is easily shown using the Myhill-Nerode theorem. Clearly, such a sequence exists if and only if the rank of $M(n)$ (as n increases) is unbounded. Moreover, the rank of $M(n)$ is unbounded if and only if the number of distinct rows in $M(n)$ is unbounded. The Myhill-Nerode theorem states that the number of equivalence classes of L (equivalently, the number of distinct rows of M) is finite if and only if L is regular. It follows that L is nonregular if and only if the rank of $M(n)$ is unbounded. This conclusion has already been noted (see sections II.3 and II.5 of [26], which describes results from the literature on rational power series and regular languages).

The above lower bound is very weak. In what follows, we significantly improve it by using the special structure of $M(n)$. Namely, we show that there is an infinite sequence of values of n such that $\text{rank}(M(n)) \geq n+1$. We define the first value of n in our sequence to be the length of the shortest word in L (clearly $\text{rank}(M(n)) \geq n+1$ in this case). To construct the remainder of the sequence, we show (in Lemma 4.9) that because L is nonregular, for any value of n, there is some $m \geq n$ such that $\text{rank}(M(n+1,m+1)) > \text{rank}(M(n,m+1))$. We then prove (in Lemma 4.10 and the proof of Theorem 4.11) that if n is such that $\text{rank}(M(n)) \geq n+1$, and we choose the smallest $m \geq n$ such that $\text{rank}(M(n+1,m+1)) > \text{rank}(M(n,m+1))$, then in fact $\text{rank}(M(m+1)) \geq m+2$.

We begin with the following useful lemma.

LEMMA 4.7. *Let $n \geq 0, m \geq 1$. Suppose that $M(n+1,m) \subseteq \text{span}(M(n,m))$. Then, for all $i, 1 \leq i \leq m$, $M(n+i, m-i+1) \subseteq \text{span}(M(n, m-i+1))$.*

Proof. The proof is by induction on i. The result is true by hypothesis of the lemma in the case $i = 1$. Suppose $1 < i \leq m$ and that the lemma is true for $i-1$.

It follows from the induction hypothesis that if $v \in M(n+i-1, m-i+2)$, then also $v \in \text{span}(M(n, m-i+2))$. Hence, it must also be the case that if $v \in M(n+i-1, m-i+1)$, then $v \in \text{span}(M(n, m-i+1))$. It remains to consider the vectors in $\hat{M}(n+i, m-i+1)$. By Fact 4.2(a), each such vector v is of the form $\text{split}^{(\sigma)}(v')$, where $v' \in M(n+i-1, m-i+2)$, for some $\sigma, |\sigma| = 1$. By the inductive hypothesis, $v' \in \text{span}(M(n, m-i+2))$. Hence, by Lemma 4.6, $v \in \text{span}(\text{split}^{(\sigma)}(M(n, m-i+2)))$.

Then, by Fact 4.2(b), all of the vectors in $\text{split}^{(\sigma)}(M(n, m-i+2))$ are in $M(n+1, m-i+1)$. Hence, $v \in \text{span}(M(n+1, m-i+1))$. Finally, by the hypothesis of the lemma, $\text{span}(M(n+1, m-i+1)) = \text{span}(M(n, m-i+1))$. Hence, $v \in \text{span}(M(n, m-i+1))$, as required. □

COROLLARY 4.8. *For any* $n \geq 0$, *if* $\operatorname{rank}(M(n+1,2p)) = \operatorname{rank}(M(n,2p)) \leq r$, *then* $\operatorname{rank}(M(p)) \leq r$.

Proof. If $n \geq p$, then $M(p)$ is a submatrix of $M(n,2p)$ so the result follows trivially. Otherwise, choose i so that $n+i=p$. Then $M(p)$ is a submatrix of $M(n+i,2p-i+1)$, and hence by Lemma 4.7, the rows of $M(p)$ are contained in $\operatorname{span}(M(n,p))$. Thus again, $\operatorname{rank}(M(p)) \leq r$. □

The following lemma shows the existence of an $m \geq n$ such that $\operatorname{rank}(M(n+1,m+1)) > \operatorname{rank}(M(n,m+1))$.

LEMMA 4.9. *Let* L *be a nonregular language. Then for any* n, *there exists an* $m \geq n$ *such that* $\operatorname{rank}(M(n+1,m+1)) > \operatorname{rank}(M(n,m+1))$.

Proof. Let r be the number of strings of length $\leq n$. Clearly, $\operatorname{rank}(M(n,m)) \leq r$ for all m, since there are r rows in $M(n,m)$. Let $p = p_r$ as in Fact 4.3, that is, $\operatorname{rank}(M(p)) \geq r+1$. Hence, by Corollary 4.8, it must be the case that $\operatorname{rank}(M(n+1,2p)) > \operatorname{rank}(M(n,2p))$. Thus, $2p$ is one possible value of m that satisfies the lemma. □

It remains to show that if n is such that $\operatorname{rank}(M(n)) \geq n+1$, and m is the smallest number such that $m \geq n$ and $\operatorname{rank}(M(n+1,m+1)) > \operatorname{rank}(M(n,m+1))$, then $\operatorname{rank}(M(m+1)) \geq m+2$. This is clearly true if for all $i \in [0,\ldots,m-n]$, $\operatorname{rank}(M(n,m-i)) < \operatorname{rank}(M(n,m-i+1))$, because in this case $\operatorname{rank}(M(n,m+1)) \geq m+2$. The difficult case is when there exist values of i such that $\operatorname{rank}(M(n,m-i)) = \operatorname{rank}(M(n,m-i+1))$. To help deal with this case, we prove the following lemma.

LEMMA 4.10. *Suppose that the following properties hold:*

1. $M(n+1,n+1) \subseteq \operatorname{span}(M(n,n+1))$.
2. m *is the smallest number* $> n$ *such that* $M(n+1,m+1) \not\subseteq \operatorname{span}(M(n,m+1))$.
3. i *is a number in the range* $[0,\ldots,m-n]$ *such that*

$$\operatorname{rank}(M(n,m-i)) = \operatorname{rank}(M(n,m-i+1)).$$

Then, there is some vector in $M(n+i+1,m-i+1)$ *which is not in* $\operatorname{span}(M(n,m-i+1))$.

Proof. Let $v' \in M(n+1,m+1) - \operatorname{span}(M(n,m+1))$, where v' is the extension of some $v \in M(n+1,m)$.

Then, we claim that for some $s, |s| = i$, $\operatorname{split}^{(s)}(v') \notin \operatorname{span}(M(n,m-i+1))$. Since $\operatorname{split}^{(s)}(v') \in M(n+i+1,m-i+1)$ by Fact 4.2(b), this is sufficient to prove the lemma.

Suppose to the contrary that for all s of length i, $\operatorname{split}^{(s)}(v') \in \operatorname{span}(M(n,m-i+1))$.

Let $\{b_1,\ldots,b_p\}$ be a basis of $M(n,m)$. Let $\{b'_1,\ldots,b'_p\}$ be an extension of this basis in $M(n,m+1)$. By properties 1 and 2 of the lemma, v is in $\operatorname{span}(M(n,m))$. Let $v = \alpha_1 b_1 + \cdots + \alpha_p b_p$. Then, applying Fact 4.1, we see that for all $s, |s| = i$,

$$\operatorname{split}^{(s)}(v) = \alpha_1 \operatorname{split}^{(s)}(b_1) + \cdots + \alpha_p \operatorname{split}^{(s)}(b_p). \tag{2}$$

We want to show that for all s of length i,

$$\operatorname{split}^{(s)}(v') = \alpha_1 \operatorname{split}^{(s)}(b'_1) + \cdots + \alpha_p \operatorname{split}^{(s)}(b'_p).$$

It follows from this and Lemma 4.5 that

$$v' = \alpha_1 b'_1 + \cdots + \alpha_p b'_p,$$

contradicting the fact that $v' \notin \operatorname{span}(M(n,m+1))$.

Consider the vectors $\text{split}^{(s)}(b'_k)$. These are in $M(n+i, m-i+1)$, by Fact 4.2 (b). If $i = 0$, this is clearly in $\text{span}(M(n, m+1))$. If $0 < i \leq m-n$, by Lemma 4.7 and by property 2 of this lemma, these vectors are in $\text{span}(M(n, m-i+1))$. Let $c_1, \ldots, c_l$ be a basis for $\text{span}(M(n, m-i))$, and for $1 \leq k \leq l$, let c'_k be an extension in $M(n, m-i+1)$ of c_k. Clearly the set $\{c'_1, \ldots, c'_l\}$ is also linearly independent, and since $\text{rank}(M(n, m-i)) = \text{rank}(M(n, m-i+1))$, this set is a basis for $\text{span}(M(n, m-i+1))$. Let

$$\text{split}^{(s)}(b'_k) = \gamma^{(s)}_{k,1} c'_1 + \cdots + \gamma^{(s)}_{k,l} c'_l. \tag{3}$$

Then, also

$$\text{split}^{(s)}(b_k) = \gamma^{(s)}_{k,1} c_1 + \cdots + \gamma^{(s)}_{k,l} c_l. \tag{4}$$

Also, since $v \in M(n+1, m)$, from Fact 4.2(b) it must be that the vectors $\text{split}^{(s)}(v)$ are in $M(n+i+1, m-i)$. Hence, again by property 2 of this lemma, and by Lemma 4.7, these vectors are in $\text{span}(M(n, m-i))$.

Since $c_1, \ldots, c_l$ is a basis for $\text{span}(M(n, m-i))$, it follows that there exists a unique sequence of coefficients $\tau_1, \ldots, \tau_l$ such that

$$\text{split}^{(s)}(v) = \tau_1 c_1 + \tau_2 c_2 + \cdots + \tau_l c_l.$$

Also, by combining equation (2) with equation (4), we see that

$$\begin{aligned} \text{split}^{(s)}(v) \quad = \quad & \alpha_1[\gamma^{(s)}_{1,1} c_1 + \cdots + \gamma^{(s)}_{1,l} c_l] \\ + \quad & \alpha_2[\gamma^{(s)}_{2,1} c_1 + \cdots + \gamma^{(s)}_{2,l} c_l] \\ + \quad & \cdots \\ + \quad & \alpha_p[\gamma^{(s)}_{p,1} c_1 + \cdots + \gamma^{(s)}_{p,l} c_l]. \end{aligned}$$

Thus $\tau_k = \alpha_1 \gamma^{(s)}_{1,k} + \cdots + \alpha_p \gamma^{(s)}_{p,k}$ for all $k \in [1, \ldots, l]$.

We claim

$$\begin{aligned} \text{split}^{(s)}(v') \quad = \quad & \alpha_1[\gamma^{(s)}_{1,1} c'_1 + \cdots + \gamma^{(s)}_{1,l} c'_l] \\ + \quad & \alpha_2[\gamma^{(s)}_{2,1} c'_1 + \cdots + \gamma^{(s)}_{2,l} c'_l] \\ + \quad & \cdots \\ + \quad & \alpha_p[\gamma^{(s)}_{p,1} c'_1 + \cdots + \gamma^{(s)}_{p,l} c'_l]. \end{aligned}$$

We now justify the claim. By our initial assumption, $\text{split}^{(s)}(v')$ is in $\text{span}(M(n, m-i+1))$. Thus for some unique coefficients $\tau'_1, \ldots, \tau'_l$,

$$\text{split}^{(s)}(v') = \tau'_1 c'_1 + \tau'_2 c'_2 + \cdots + \tau'_l c'_l.$$

Each c'_k is an extension of c_k, and there is a unique linear combination of $c_1, c_2, \ldots c_l$ that is equal to $\text{split}^{(s)}(v)$. It follows that each $\tau'_k = \tau_k$. This proves the claim.

Combining the claim with equation (3) yields

$$\text{split}^{(s)}(v') = \alpha_1 \text{ split}^{(s)}(b'_1) + \cdots + \alpha_p \text{ split}^{(s)}(b'_p),$$

as desired. □

We now prove the lower bound.

THEOREM 4.11. *If L is nonregular, then* $\text{rank}(M(n)) \geq n+1$ *infinitely often.*

Proof. The base case is n such that the shortest word in the language is of length n.

Suppose that $\text{rank}(M(n)) \geq n+1$ for some fixed n. Let m be the smallest number $\geq n$ such that $\text{rank}(M(n+1,m+1)) > \text{rank}(M(n,m+1))$. By Lemma 4.9 there is such an m. We claim that $\text{rank}(M(m+1)) \geq m+2$.

If $m = n$, then the claim is clearly true. Suppose $m > n$.

Let B_k be a basis for $M(n,k)$, $n \leq k \leq m+1$, where the extensions of all vectors in B_k are in B_{k+1}. Let B'_{k-1} denote the subset of B_k which are extensions of vectors in B_{k-1}.

We construct a set of $m+2$ linearly independent vectors in $M(m+1)$ as follows. For k from n to $m+1$, we define a linearly independent set C_k of vectors in $M(m+1,k)$, of size at least $k+1$. Then, C_{m+1} is the desired set.

Let $C_n = B_n$. This is by definition a linearly independent set, and it has size $\geq n+1$ because (by our initial assumption) $\text{rank}(M(n)) \geq n+1$. Suppose that $n \leq k < m+1$ and that C_k is already constructed and is linearly independent. Construct C_{k+1} as follows.

(i) Let C'_k be the set of extensions in $M(m+1,k+1)$ of the vectors in C_k. Add C'_k to C_{k+1}.

(ii) Add B_{k+1} to C_{k+1}. (Thus, C_{k+1} is expanded to contain those vectors in B_{k+1} which are not in B'_k.)

(iii) Finally, suppose nothing is added to C_{k+1} in step (ii); that is, $\text{rank}(M(n,k)) = \text{rank}(M(n,k+1))$. If i is such that $k = m-i$, then this is equivalent to $\text{rank}(M(n,m-i)) = \text{rank}(M(n,m-i+1))$. Thus, we can apply Lemma 4.10 to obtain a vector $v' \in M(n+i+1, m-i+1)$ which is not in $\text{span}(M(n,m-i+1))$. (Thus, $v' \in M(n+m+1-k, k+1)$ but is not in $\text{span}(B'_k)$.) Add v' to C_{k+1}.

We claim that the vectors in C_{k+1} are linearly independent. Clearly the set C'_k is linearly independent. Consider each vector u' added to C_{k+1}, which is not in C'_k. By the construction, u' is not in $\text{span}(B'_k)$. Let u' be the extension of vector u in $M(m+1,k)$. We claim that the vector u must be linearly dependent on the set B_k. This is true if u' is added in step (ii), since in this case u is in $M(n,k)$ and B_k is a basis for $M(n,k)$. It is also true in the case that $u' = v'$, the vector added in step (iii), since then by Lemma 4.7, $u = v \in \text{span}(B_k)$.

Hence, $u \in \text{span}(C_k)$, since $B_k \subseteq C_k$. Moreover, u can be expressed as a unique linear combination of the vectors of C_k, with nonzero coefficients only on those vectors in B_k.

If u' were in $\text{span}(C'_k)$, then since it is an extension of u, it would also be expressible as a unique linear combination of the vectors of C'_k, with nonzero coefficients only on those vectors in B'_k. But that contradicts the fact that $u' \notin \text{span}(B'_k)$. □

4.2.3. The tiling complexity lower bound.

THEOREM 4.12. *If L is nonregular, then the 1-tiling complexity of either L or $\bar{L}$ is at least $2^{\sqrt{\log n}-2} - 1$ infinitely often.*

Proof. Melhorn and Schmidt, and, independently, Orlin, showed that for any binary matrix A, *rank*$(A) \leq \tilde{T}(A)$ [21, 22]. Their result holds for A over any field. Halstenberg and Reischuk [13], refining a proof of Aho, Ullman, and Yannakakis [1], showed that $\lceil \log \tilde{T}(A) \rceil \leq \lceil \log T^1(A) \rceil (\lceil \log(T^0(A)+1) \rceil + 2) + 1$. Let $T^*(A) = \max(T^1(A), T^0(A))$. Then $\lceil \log \text{rank}(A) \rceil \leq (\lceil \log(T^*(A)+1) \rceil + 1)^2$.

By Theorem 4.11, if L is nonregular, then the rank of $M(n)$ is at least $n+1$ infinitely often. It follows that for infinitely many n, $T^*(M(n)) = \max(T^1_L(n), T^0_L(n)) \geq 2^{\sqrt{\log n}-2} - 1$. □

5. Variations on the model. In this section, we discuss extensions of our main results to other related models.

We first show that Theorem 1.1 also holds for the following "alternating probabilistic" finite state automaton model. In this model, which we call a 2apfa, the nondeterministic states N are partitioned into two subsets N_E and N_U of existential and universal states, respectively. Accordingly, for a fixed input, there are two types of strategy, defined as follows for a fixed input string $w = w_0 w_1 w_2 \dots w_n w_{n+1}$. An *existential (universal) strategy* on w is a function

$$E_w : N_E \times \{0, \dots, n+1\} \to Q \times \{-1, 0, 1\},$$

$$(U_w : N_U \times \{0, \dots, n+1\} \to Q \times \{-1, 0, 1\}),$$

such that $\delta(q, \sigma, q', d) = 1$ whenever $E_w(q, j) = (q', d)$ $(U_w(q, j) = (q', d))$ and $w_j = \sigma$.

A language $L \subseteq \Sigma^*$ is accepted with *bounded error probability* if for some constant $\epsilon < 1/2$,

1. for all $w \in L$, there exists an existential strategy E_w on which the automaton accepts with probability $\geq 1 - \epsilon$ on all universal strategies U_w, and
2. for all $w \notin L$, on every existential strategy E_w, the automaton accepts with probability $\leq \epsilon$ on some universal strategy U_w.

The complexity classes 1APFA, 1APFA-polytime, and so on, are defined in the natural way, following our conventions for the npfa model.

THEOREM 5.1. *1APFA = Regular.*

Proof. As in Theorems 1.1 and 3.1, we show that if L is a language accepted by a 1APFA, then the tiling complexity of L is bounded. We first extend the notation of Theorem 3.1.

If E is an existential strategy on xy and U is a universal strategy on xy, let $\mathbf{p}_{xy}(E, U)$ be the state probability (row) vector at the step when the input head moves off the right end of x, on the strategies E, U. Let $\mathbf{r}_{xy}(E, U)$ be the column vector whose ith entry is the probability of accepting the string xy, assuming that the automaton is in state i at the moment that the head moves off the right end of x, on the strategies E, U. For each 1-entry $[x, y]$ of M_L, fix an existential strategy E_{xy}, that causes xy to be accepted with probability at least $1 - \epsilon$, for all universal strategies.

Partition the space $[0, 1]^c$ into cells of size $\mu \times \mu \times \cdots \times \mu$, as before. Let $\mathcal{C}$ be a nonempty subset of the cells. We say that entry $[x, y]$ of M_L belongs to $\mathcal{C}$ if $xy \in L$, and $\mathcal{C}$ is the smallest set of cells which contain all the vectors $\mathbf{p}_{xy}(E_{xy}, U)$, for all universal strategies U.

With each nonempty subset $\mathcal{C}$ of the cells, associate a rectangle $R_{\mathcal{C}}$ defined as follows.

$$\{x \mid \text{there exists } y \text{ such that } [x, y] \text{ belongs to } \mathcal{C}\}$$
$$\times$$
$$\{y \mid \text{there exists } x \text{ such that } [x, y] \text{ belongs to } \mathcal{C}\}.$$

Then, $R_{\mathcal{C}}$ is a valid 1-tile. To see this, suppose that $[x, y] \in R_{\mathcal{C}}$. If $[x, y]$ belongs to $\mathcal{C}$, then it must be a 1-entry. Otherwise, there exist x' and y' such that $[x, y']$ and $[x', y]$ belong to $\mathcal{C}$.

Consider the strategy E that, while reading x, uses the strategy $E_{xy'}$, and while reading y, uses the strategy $E_{x'y}$. We claim that xy is accepted with probability at least $1/2$ on existential strategy E and any universal strategy U on xy. The probability that xy is accepted on strategies E, U is

$$\mathbf{p}_{xy}(E, U)\mathbf{r}_{xy}(E, U) = \mathbf{p}_{xy'}(E_{xy'}, U)\mathbf{r}_{x'y}(E_{x'y}, U).$$

Since $[x, y']$ and $[x', y]$ belong to the same set of cells $\mathcal{C}$, $\mathbf{p}_{xy'}(E_{xy'}, U)$ and $\mathbf{p}_{x'y}(E_{x'y}, U')$ are in the same cell, for some universal strategy U'. Moreover,

$$\mathbf{p}_{x'y}(E_{x'y}, U')\mathbf{r}_{x'y}(E_{x'y}, U) \geq 1 - \epsilon.$$

This is because this quantity is the probability that $x'y$ is accepted on existential strategy $E_{x'y}$ and a universal strategy which is a hybrid of U and U'; also, by definition of $E_{x'y}$, the probability that $x'y$ is accepted with respect to $E_{x'y}$ and any universal strategy is $\geq 1 - \epsilon$. Hence,

$$\begin{aligned}(\mathbf{p}_{x'y}(E_{x'y}, U') - \mathbf{p}_{xy'}(E_{xy'}, U))\ \mathbf{r}_{x'y}(E_{x'y}, U) & \\ &= \textstyle\sum_{i=1}^{c} [\mathbf{p}_{x'y}(E_{x'y}, U') - \mathbf{p}_{xy'}(E_{xy'}, U)]_i [\mathbf{r}_{x'y}(E_{x'y}, U)]_i \\ &\leq \mu \textstyle\sum_{i=1}^{c} [\mathbf{r}_{x'y}(E_{x'y}, U)]_i \\ &\leq \mu c \\ &= 1/2 - \epsilon, \quad \text{by our choice of } \mu.\end{aligned}$$

Hence, the probability that xy is accepted on the strategies E, U is

$$\begin{aligned}\mathbf{p}_{xy'}(E_{xy'}, U)\mathbf{r}_{x'y}(E_{x'y}, U) &\geq \mathbf{p}_{x'y}(E_{x'y}, U')\mathbf{r}_{x'y}(E_{x'y}, U) - (1/2 - \epsilon) \\ &\geq (1 - \epsilon) - (1/2 - \epsilon) \\ &= 1/2 > \epsilon.\end{aligned}$$

Since U is arbitrary, it follows that there is an existential strategy E such that on all strategies U, the probability that xy is accepted on the strategies E, U is greater than ϵ, and so it cannot be that $xy \notin L$. Hence, for all $[x, y] \in R_{\mathcal{C}}$, xy must be in L. Therefore $R_{\mathcal{C}}$ is a 1-tile in M_L.

The proof is completed as in Theorem 3.1. □

In the same way, Theorem 3.4 can also be extended to obtain the following.

THEOREM 5.2. *A language L is in 2APFA-polytime only if the 1-tiling complexity of L is bounded by $2^{polylog(n)}$.*

Thus, for example, the language *Pal*, consisting of all strings over $\{0, 1\}^*$ which read the same forwards as backwards, is not in the class 2APFA-polytime. To see this, consider the submatrix of $M_L(n)$, consisting of all rows and columns labeled by strings of length exactly n. This matrix contains a fooling set of size 2^n; hence a 1-tiling of $M_L(n)$ requires at least 2^n tiles.

We next extend Theorem 1.2 to automata with $o(\log \log n)$ space. We refer to these as Arthur–Merlin games, since this is the usual notation for such automata which are not restricted to a finite number of states [7]. The definition of an Arthur–Merlin game is similar to that of an npfa, except that the machine has a fixed number of read/write worktapes. The Arthur–Merlin game runs within space $s(n)$ if on any input w with $|w| \leq n$, at most $s(n)$ tape cells are used on any worktape. Thus, the number of different configurations of the Arthur–Merlin game is $2^{O(s(n))}$.

THEOREM 5.3. *Let M and $\bar{M}$ be Arthur–Merlin games which recognize a nonregular language L and its complement $\bar{L}$, respectively, within space $o(\log \log n)$. Suppose that the expected running time of both M and $\bar{M}$ is bounded by $t(n)$. Then, for all $b < 1/2$, $\log \log t(n) \geq (\log n)^b$. In particular, $t(n)$ is not bounded by any polynomial in n.*

Proof. The proof of Theorem 1.2 can be extended to space bounded Arthur–Merlin games to yield the following generalization of equation (1). Let $c(n)$ be an upper bound on the number of different configurations of M on inputs of length n,

and let $d(n) = 2c(n) + 4$. Then, for sufficiently large n, the number of 1-tiles needed to cover $M_L(n)$ is at most

$$T_L^1(n) \leq (\lceil 2 \log_2 t(n)/\mu \rceil + 1)^{d^2(n)} = 2^{\Theta(d^2(n) \log\log t(n))}.$$

Since M uses $o(\log\log n)$ space, for any constant $c > 0$, $d(n) \leq (\log n)^c$, for sufficiently large n.

Now, suppose to the contrary that for some $b < 1/2$, $\log\log t(n) < (\log n)^b$ for sufficiently large n. Then,

$$d^2(n) \log\log t(n) = o(\sqrt{\log n}).$$

Hence, the number of tiles needed to cover the 1-valued entries of $M_L(n)$ is $2^{o(\sqrt{\log n})}$. The same argument for $\bar{M}$ shows that also, for sufficiently large n, the number of tiles needed to cover the 1-valued entries of $M_{\bar{L}}(n)$ is $2^{o(\sqrt{\log n})}$.

Hence, by Theorem 4.12 L must be regular, which is a contradiction. □

Finally, we consider a restriction of the 2npfa model, which, given polynomial time, can only recognize regular languages. A *restricted 2npfa* is a 2npfa for which there is some $\epsilon < 1/2$ such that on all inputs w and strategies S_w, the probability that the automaton accepts is either $\geq 1 - \epsilon$ or $< \epsilon$.

THEOREM 5.4. *Any language accepted by a restricted 2npfa with bounded error probability in polynomial time is regular.*

Proof. Let L be accepted by a 2npfa M with bounded error probability in polynomial expected time. Let Σ be the alphabet, δ the transition function, $Q = \{q_1, q_2, \ldots, q_{|Q|}\}$ the set of states, and $N \subset Q$ the set of nondeterministic states of M. Without loss of generality, let $N = \{q_1, \ldots, q_{|N|}\}$.

We first define a representation of strategies as strings over a finite alphabet. Let $\Sigma' = (N \times Q \times \{-1, 0, 1\})^{|N|}$. Without loss of generality, assume that $\Sigma \cap \Sigma' = 0$. A string $S_0 S_1 \ldots S_{n+1}$ corresponds to a strategy on $\mathfrak{c}\, w\$$, where $\mathfrak{c}\, w\$ = \sigma_0 \sigma_1 \ldots \sigma_{n+1}$, if for $0 \leq j \leq n+1$, S_j is of the form

$$S_j = ((q_1, q_1', d_1), (q_2, q_2', d_2), \ldots, (q_{|N|}, q_{|N|}' d_{|N|})),$$

and $\delta(q_i, \sigma_j, q_i', d_i) = 1$.

Define L' to be the set of strings of the form $\sigma_0 S_0 \sigma_1 S_1 \ldots \sigma_{n+1} S_{n+1}$, where each σ_i is in the alphabet Σ, each S_i is in the alphabet Σ', and furthermore, $S = S_0 S_1 \ldots S_{n+1}$ corresponds to a strategy of M on input $w = \sigma_0 \sigma_1 \ldots \sigma_{n+1}$, which causes w to be accepted.

Then, L' is accepted by a 2pfa with bounded error probability in polynomial time. Thus, L' is regular [7]. Moreover, note that a string of the form $w = \sigma_0 \sigma_1 \ldots \sigma_{n+1}$ is in L if and only if for some choice of $S_0, S_1 \ldots S_{n+1}$, $\sigma_0 S_0 \sigma_1 S_1 \ldots \sigma_{n+1} S_{n+1}$ is in L'. Let M' be a one-way deterministic finite state automaton for L' and assume, without loss of generality, that the set of states in which M' can be when the head is at an even position is disjoint from the set of states in which M' can be when the head is at an odd position. Then, from M' we can construct a one-way nondeterministic finite state automaton for L by replacing the even position states by nondeterministic states. Hence, L is regular. □

6. Conclusions. We have introduced a new measure of the complexity of a language, namely its tiling complexity, and have proved a gap between the tiling complexity of regular and nonregular languages. We have applied these results to prove limits on the power of finite state automata with both probabilistic and nondeterministic states. These results first appeared in [5].

An intriguing question left open by this work is whether the class 2NPFA-polytime is closed under complement. If it is, we can conclude that 2NPFA-polytime = Regular. Recall that the class 2NPFA does contain nonregular languages, since it contains the class 2PFA, and Freivalds [10] showed that $\{0^n1^n \mid n \geq 0\}$ is in this class. However, Kaņeps [18] showed that the class 2PFA does not contain any nonregular unary language. Another open question is whether the class 2NPFA contains any nonregular unary language. It is also open whether there is a nonregular language in 2APFA-polytime.

There are several other interesting open problems. Can one obtain a better lower bound on the tiling complexity of nonregular languages than that given by Theorem 4.12, perhaps by an argument that is not based on rank? We know of no nonregular language with tiling complexity less than $\Omega(n)$ infinitely often, so the current gap is wide.

REFERENCES

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

[2] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and hardness of approximation problems*, in Proc. 33rd IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 14–23.

[3] L. Babai and S. Moran, *Arthur-Merlin games: A randomized proof system and a hierarchy of complexity classes*, J. Comput. System Sci., 36 (1988), pp. 254–276.

[4] A. Condon, *Computational Models of Games*, MIT Press, Cambridge, MA, 1989.

[5] A. Condon, L. Hellerstein, S. Pottle, and A. Wigderson, *On the power of finite automata with both nondeterministic and probabilistic states*, in Proc. 26th Annual ACM Symposium on Theory of Computing, ACM, New York, 1994, pp. 676–685.

[6] A. Condon and R. Ladner, *Probabilistic game automata*, J. Comput. System Sci., 36 (1988), pp. 452–489.

[7] C. Dwork and L. Stockmeyer, *A time-complexity gap for two-way probabilistic finite state automata*, SIAM J. Comput., 19 (1990), pp. 1011–1023.

[8] C. Dwork and L. Stockmeyer, *Finite state verifiers* I*: The power of interaction*, J. Assoc. Comput. Mach., 39 (1992), pp. 800–828.

[9] L. Fortnow and C. Lund, *Interactive proof systems and alternating time-space complexity*, in Theoret. Comput. Sci., 113 (1993), pp. 55–73.

[10] R. Freivalds, *Probabilistic two-way machines*, in Proc. International Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 188, Springer-Verlag, New York, 1981, pp. 33–45.

[11] G. Frobenius, *Über das Trägheitsgesetz der quadratischen Formen*, Sitzungsber. der Königl. Preuss. Akad. der Wiss., 1894, pp. 407–431.

[12] A. G. Greenberg and A. Weiss, *A lower bound for probabilistic algorithms for finite state machines*, J. Comput. System Sci., 33 (1986), pp. 88–105.

[13] B. Halstenberg and R. Reischuk, *On different modes of communication*, in Proc. 20th Annual ACM Symposium on the Theory of Computing, ACM, New York, 1988, pp. 162–172.

[14] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison–Wesley, Reading, MA, 1979.

[15] I. S. Iohvidov, *Hankel and Toeplitz Matrices and Forms: Algebraic Theory*, I. Gohberg, ed., G. Philip, and A. Thijsse, translators, Birkhäuser, Boston, 1982.

[16] J. Kaņeps and R. Freivalds, *Minimal nontrivial space complexity of probabilistic one-way Turing machines*, in Proc. Conference on Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 452, Springer-Verlag, New York, 1990, pp. 355–361.

[17] J. Kaņeps and R. Freivalds, *Running time to recognize nonregular languages by* 2*-way probabilistic automata*, in Proc. 18th International Colloquium on Automata, Languages, and Programming, Springer-Verlag, New York, 1991, pp. 174-185.

[18] J. Kaņeps, *Regularity of one-letter languages acceptable by* 2*-way finite probabilistic automata*, in Proc. Fundamentals of Computation Theory, Lecture Notes in Comput. Sci. 529, Springer-Verlag, New York, 1991, pp. 287–296.

[19] R. M. Karp, *Some bounds on the storage requirements of sequential machines and Turing machines*, J. Assoc. Comput. Mach., 14 (1967), pp. 478–489.
[20] F. T. Leighton and R. L. Rivest, *The Markov Chain Tree Theorem*, Tech. Report MIT/LCS/TM-249, Laboratory for Computer Science, MIT, Cambridge, MA, 1983; also in IEEE Trans. Inform. Theory, IT-37 (1986), pp. 733–742.
[21] K. Melhorn and E. M. Schmidt, *Las Vegas is better than determinism in VLSI and distributed computing*, in Proc. 14th Annual ACM Symposium on Theory of Computing, ACM, New York, 1982, pp. 330–337.
[22] J. Orlin, *Contentment in graph theory: Covering graphs with cliques*, in Proc. Koniklijke Nederlandse Akademie van Wetenschappen, Amsterdam Series A, 80 (1977), pp. 406–424.
[23] C. Papadimitriou, *Games against nature*, J. Comput. System Sci., 31 (1985), pp. 288–301.
[24] M. O. Rabin, *Probabilistic automata*, Inform. Control, 6 (1963), pp. 230–245.
[25] M. O. Rabin and D. Scott, *Finite automata and their decision problems*, IBM J. Research, 3 (1959), pp. 115–125.
[26] A. Salomaa and M. Soittola, *Automata-Theoretic Aspects of Formal Power Series*, Texts and Monographs in Computer Science, Springer-Verlag, New York, 1978.
[27] E. M. Schmidt, *Succinctness of Description of Context Free, Regular and Unambiguous Languages*, Ph.D. thesis, Cornell University, Ithaca, NY, 1978.
[28] J. Shallit, *Automaticity:* I. *Properties of a measure of descriptional complexity*, J. Comput. System Sci., 53 (1996), pp. 10–25.
[29] A. C. Yao, *Some complexity questions related to distributed computing*, in Proc. 11th Annual ACM Symposium on Theory of Computing, ACM, New York, 1979, pp. 209–213.
[30] A. C. Yao, *Lower bounds by probabilistic arguments*, in Proc. 24th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1983, pp. 420–428.

SIAM J. COMPUT.
Vol. 27, No. 3, pp. 763–803, June 1998
© 1998 Society for Industrial and Applied Mathematics
011

A PARALLEL REPETITION THEOREM*

RAN RAZ†

Abstract. We show that a parallel repetition of any two-prover one-round proof system (MIP(2, 1)) decreases the probability of error at an exponential rate. No constructive bound was previously known. The constant in the exponent (in our analysis) depends only on the original probability of error and on the total number of possible answers of the two provers. The dependency on the total number of possible answers is logarithmic, which was recently proved to be almost the best possible [U. Feige and O. Verbitsky, *Proc. 11th Annual IEEE Conference on Computational Complexity*, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 70–76].

Key words. interactive proofs, parallel repetition, direct product

AMS subject classifications. 68Q25, 68R99

PII. S0097539795280895

1. Introduction and basic notations.

1.1. History and motivation. We consider two-prover one-round proof systems (MIP(2, 1)), as introduced in [12]. In an MIP(2, 1) proof system, two computationally unbounded provers (that are not allowed to communicate with each other) try to convince a probabilistic polynomial time verifier that a common input I belongs to a prespecified language L. The proof proceeds in one round: The verifier generates a pair of questions (x, y), based on I and on a random string r, and sends x to the first prover and y to the second prover. The first prover responds by sending u, and the second prover responds by sending v. Based on I, r, x, y, u, v, the verifier decides whether to accept or reject the conjecture: $I \in L$. In what follows, the strategy of the verifier (the way to generate the questions and to decide whether $I \in L$) is called "a proof system," and the pair of strategies of the provers is called "a protocol." A language L is in MIP(2, 1) with probability of error ϵ if there exists a proof system (of this type), such that: (1) For $I \in L$, there exists a protocol that causes the verifier to always accept; (2) For $I \notin L$, for any possible protocol, the verifier accepts with probability smaller than ϵ. For the exact definitions of MIP(2, 1) proof systems and the family of languages MIP(2, 1), see [12, 23].

The class of languages MIP(2, 1) turned out to be very powerful. In particular, it follows from [7, 37, 25] that NEXPTIME = MIP(2, 1) with exponentially small probability of error. MIP(2, 1) proof systems have cryptographic applications (see [12, 13, 36, 18]), and have also been used as a starting point to prove that certain optimization problems are hard to approximate (see [22, 4, 5, 25, 6, 10, 38, 8]). For more discussion about the class MIP(2, 1) and its applications, see [23] and the references there.

Sequential repetition of MIP(2, 1) proof systems decreases the probability of error exponentially, but requires multiple rounds. Parallel repetition preserves the number of rounds. At what rate is the probability of error decreased by parallel repetition? At first, it was believed that, as in the sequential case, repeating a proof system k times

*Received by the editors January 17, 1995; accepted for publication (in revised form) July 29, 1996. A preliminary version of the paper appeared in Proceeding of the 27th Annual ACM Symposium on the Theory of Computing, ACM, New York, 1995, pp. 447–556.

http://www.siam.org/journals/sicomp/27-3/28089.html

†Department of Applied Mathematics, Weizmann Institute, Rehovot 76100, Israel (ranraz@wisdom.weizmann.ac.il).

in parallel decreases the probability of error to ϵ^k (see [26, 28]). A counter example for this conjecture was given in [21] (see also [36, 19, 25, 27]). For years it was not even known whether parallel repetition can make the probability of error arbitrarily small. This was recently proved in [44]. No constructive bound, however, was given there for the number of repetitions required to decrease the probability of error below a given bound.

In the simpler special case, where the provers' questions, x and y, are chosen independently, it has been known that repeating a proof system k times in parallel decreases the probability of error exponentially. The proof was first given in [15], and the bound was further improved in [36, 39, 19, 1]. Another special case, a tree-like-measure case, was recently settled in [45]. The constant in the exponent in all those proofs, however, is (at least) polynomial in the number of possible questions (x, y). We remark that in most interesting cases, the questions x, y are not chosen independently and the measure is not tree-like.

Researchers were interested in analyzing the probability of error of a parallel repetition, not only as a mathematical problem, but also because an efficient technique to decrease the probability of error was needed. In the literature there are many results that use different techniques to decrease the probability of error [19, 33, 37, 25, 8, 23, 43]. For certain applications, however, these techniques are insufficient. Parallel repetition was suggested as a technique to decrease the probability of error, because it was believed to be very efficient, and because it preserves many canonical properties of the proof system (e.g., zero knowledge).

Since the appearance of a preliminary version of this paper in STOC95 [40], our results were used by [9, 32] to improve many of the hardness results for approximation of optimization problems (see also [2, 3, 24, 11, 30, 31]). In particular, [32] uses our results to prove very impressive optimal inapproximability results for some of the most basic optimization problems (e.g., for Max-3-SAT). Our result was also used by [42] to prove a new direct sum theorem for probabilistic communication complexity.

The reader can find more about the problem of parallel repetition in [19, 25, 23, 20].

1.2. Main result. We will concentrate on the deterministic case, where the verifier decides whether to accept or reject the conjecture: $I \in L$, based on I, x, y, u, v only (and not on the random string r). The probabilistic case, where this decision depends also on r, is shortly discussed in section 7.

Following [15, 19, 25], we model the problem as a problem on games:
A game G consists of four finite sets X, Y, U, V, with a probability measure $\mu : X \times Y \to \mathbf{R}^+$, and a predicate $Q : X \times Y \times U \times V \to \{0, 1\}$. We think of Q also as a set. A protocol for G consists of a function $u : X \to U$, and a function $v : Y \to V$. The value of the protocol is defined as

$$\sum_{X \times Y} \mu(x, y) Q(x, y, u(x), v(y)),$$

i.e., the μ-probability that $Q(x, y, u(x), v(y)) = 1$. The value of the game, $w(G)$, is defined to be the maximal value of all protocols for G. The answer size of the game, $s(G)$, is defined by $s(G) = |U||V|$.

We think of G as a game for two players (provers): Player I receives $x \in X$, and player II receives $y \in Y$, according to the pair distribution $\mu(x, y)$. We think of x, y as the inputs for the game. Each player doesn't know the other player's input. Player I has to give an "answer" $u' \in U$, and player II has to give an "answer" $v' \in V$.

The goal of the players is to maximize the μ-probability that Q is satisfied (i.e., the probability that $Q(x, y, u', v') = 1$). We remark that this probability corresponds to the probability of error.

The game $G \otimes G$ consists of the sets $X \times X$, $Y \times Y$, $U \times U$, $V \times V$, with the measure

$$\mu \otimes \mu((x_1, x_2), (y_1, y_2)) = \mu(x_1, y_1)\mu(x_2, y_2)$$

and the predicate

$$Q \otimes Q((x_1, x_2), (y_1, y_2), (u_1, u_2), (v_1, v_2)) = Q(x_1, y_1, u_1, v_1)Q(x_2, y_2, u_2, v_2).$$

In the same way, the game $G^{\otimes k}$ consists of the sets X^k, Y^k, U^k, V^k, with the measure

$$\mu^{\otimes k}(x, y) = \prod_{i=1}^{k} \mu(x_i, y_i)$$

and the predicate

$$Q^{\otimes k}(x, y, u, v) = \prod_{i=1}^{k} Q(x_i, y_i, u_i, v_i),$$

where $x \in X^k, y \in Y^k, u \in U^k, v \in V^k$. Here and throughout, z_i stands for the ith coordinate of a vector z. We denote $G^{\otimes k}, Q^{\otimes k}, X^k, Y^k, U^k, V^k, \mu^{\otimes k}$ also by $\bar{G}, \bar{Q}, \bar{X}, \bar{Y}, \bar{U}, \bar{V}, \bar{\mu}$.

Assume w.l.o.g. that $s(G) \geq 2$. In this paper we prove the following parallel repetition theorem.

THEOREM 1.1. *There exists a global function $W : [0, 1] \to [0, 1]$, with $z < 1 \Rightarrow W(z) < 1$, such that given a game G, with value $w(G)$, and answer size $s(G) \geq 2$:*

$$w(G^{\otimes k}) \leq W(w(G))^{k/\log_2(s(G))}.$$

The exact behavior of the function $W(z)$ is not the focus of this paper. We will make, however, a few comments about the function $W(z)$ implicit in the paper:

1. When z tends to 0, the function $W(z)$ (implicit in this paper) tends to a constant $const_1 > 0$. In fact, for every $0 < z \leq 1$, $W(z) > const_1$. It is still not clear whether a tendency of $W(z)$ to 0, when z tends to 0, can be achieved.
2. Obviously, if $w(G)$ is a global constant (e.g., $w(G) = 1/2$) then $W(w(G))$ is just a different global constant. For example, there exists a constant $const_2 < 1$, s.t. for every $0.01 \leq z \leq 0.99$, $const_1 < W(z) < const_2$.
3. Obviously, when z tends to 1, $W(z)$ also tends to 1. It will be simpler to denote $t = 1 - z$, and to analyze the behavior of $[1 - W(1 - t)]$ when t tends to 0. It is implicit in this paper that there exists a (small) constant $const_3$, (1/34 seems to be enough), s.t. when t tends to 0, $[1 - W(1 - t)]$ can be bounded by $O(t^{const_3})$. For example, there exists a constant $const_4$, s.t. for every $t \leq 0.01$, $[1 - W(1 - t)] \leq const_4 \cdot t^{const_3}$, i.e.,

$$W(1 - t) \geq 1 - const_4 \cdot (1 - z)^{const_3}.$$

It was recently shown in [27] that in certain examples the number of repetitions, k, required to decrease the probability of error from $w(G) = 1/2$ to $w(G^{\otimes k}) \leq 1/8$ is

$$\Omega\left(\frac{\log_2(s(G))}{\log_2 \log_2(s(G))}\right).$$

This shows that the factor $\log_2(s(G))$ in Theorem 1.1 is almost the best possible.

It was observed by [42] that in Theorem 1.1, the term $\log_2(s(G))$ can be replaced with the (possibly smaller) $CC(G)$, or with the (possibly smaller) $\rho(G)$, defined in the following way.

For every $(x, y) \in X \times Y$, define $Q_{x,y} : U \times V \to \{0, 1\}$, by $Q_{x,y}(u, v) = Q(x, y, u, v)$. Define $CC_{x,y}$ to be the deterministic communication complexity of the function $Q_{x,y}$, and define $\rho_{x,y}$ to be the exact cover number of the same function (for the definitions see [42, 34]). Then define $CC(G)$ to be the maximum, taken over x, y, of $CC_{x,y}$, and define $\rho(G)$ to be the maximum, taken over x, y, of $\rho_{x,y}$. It is well known (and very easy to prove) that

$$\rho(G) \leq CC(G) \leq \log_2(s(G)).$$

Throughout the paper (sections 2, 3, 4, 5, 6), X, Y, U, V, Q, μ refer to the game G, from Theorem 1.1. Similarly, $k, \bar{G}, \bar{X}, \bar{Y}, \bar{U}, \bar{V}, \bar{Q}, \bar{\mu}$ refer to Theorem 1.1 as well. Denote, for the rest of the paper $s = s(G) = |U||V|$.

1.3. Main technical theorem. In what follows, we will sometimes denote the game G by G_μ, and denote its value by $w(\mu)$. We will sometimes say that the protocol is a protocol for μ (as X, Y, U, V, Q will be fixed). We will look at the values $w(\gamma)$, for different probability measures $\gamma : X \times Y \to \mathbf{R}^+$.

For a measure $\alpha : \Omega \to \mathbf{R}^+$, and a set $C \subset \Omega$, we use the usual notation

$$\alpha(C) = \sum_{z \in C} \alpha(z).$$

For a probability measure $\alpha : \Omega \to \mathbf{R}^+$, and a set $C \subset \Omega$, denote by $\alpha_C : \Omega \to \mathbf{R}^+$ the probability measure

$$\alpha_C(z) = \begin{cases} 0 & \text{for } z \notin C, \\ \frac{\alpha(z)}{\alpha(C)} & \text{for } z \in C. \end{cases}$$

We will think of α_C also as $\alpha_C : C \to \mathbf{R}^+$. This definition makes sense only if $\alpha(C) > 0$. For C, with $\alpha(C) = 0$, define $\alpha_C(z)$ to be identically 0.

More generally, the term $\frac{0}{0}$ can appear in some places in this paper. Unless said otherwise, $\frac{0}{0}$ is defined to be 0. The term $0z$ is defined to be 0, even if $z = \infty$, or z is undefined. This can occur when we take the expectancy (or a weighted average) of a variable z, which is undefined with probability 0 (or with a weight of 0).

For a set $C \subset \bar{X} \times \bar{Y}$, define the game $G_C^{\otimes k} = \bar{G}_C$ to consist of $\bar{X}, \bar{Y}, \bar{U}, \bar{V}$, with the measure $\bar{\mu}_C$ and the predicate $\bar{Q}$. We think of $\bar{G}_C$ as the restriction of $\bar{G}$ to the set C.

In the proof we will always work with a product set $A = A_X \times A_Y$, where $A_X \subset \bar{X}$, $A_Y \subset \bar{Y}$. Since in most parts of the paper we work with one specific $A = A_X \times A_Y$, it will be convenient to denote (throughout the paper) the measure

$\bar{\mu}_A$ by $\bar{\pi}$. The game $\bar{G}_A$ will also be denoted by $\bar{G}_{\bar{\pi}}$, and its value will also be denoted by $\bar{w}(\bar{\pi})$.

Define the predicate $\bar{Q}^i : \bar{X} \times \bar{Y} \times U \times V \to \{0,1\}$ by

$$\forall x \in \bar{X}, y \in \bar{Y}, u' \in U, v' \in V \ : \ \bar{Q}^i(x,y,u',v') = Q(x_i, y_i, u', v').$$

Define the game $\bar{G}^i_A$ to consist of $\bar{X}, \bar{Y}, U, V$ with the measure $\bar{\pi}$, and the predicate $\bar{Q}^i$. We also denote $\bar{G}^i_A$ by $\bar{G}^i_{\bar{\pi}}$, and its value by $w_i(\bar{\pi})$. We think of $\bar{G}^i_{\bar{\pi}}$ as the restriction of $\bar{G}_{\bar{\pi}}$ to one coordinate. Notice that for an input $(x,y) \in \bar{X} \times \bar{Y}$, the game $\bar{G}_{\bar{\pi}}$ just means playing simultaneously all the games $\bar{G}^i_{\bar{\pi}}$ (on the same input).

The following is our main technical theorem. The theorem claims that if $A = A_X \times A_Y$ is large, then $w_i(\bar{\pi})$ is small for at least one coordinate i.

THEOREM 1.2. *There exists a global function $W_2 : [0,1] \to [0,1]$, with $z < 1 \Rightarrow W_2(z) < 1$, and a global constant c_0, such that for all games G: for any k, and any product set $A = A_X \times A_Y \subset \bar{X} \times \bar{Y}$ (where $A_X \subset \bar{X}, A_Y \subset \bar{Y}$), and any $0 \le \Delta \le 1$, if $-\log_2 \bar{\mu}(A) \le \Delta k$ (i.e., $\bar{\mu}(A) \ge 2^{-\Delta k}$) then there exists i with*

$$w_i(\bar{\pi}) \le W_2(w(G)) + c_0 \Delta^{1/16}.$$

Again, achieving the best function W_2 and the best constant c_0, and improving the constant $1/16$, are not the focus of this paper.

1.4. More notations and basic facts. For a measure or function $\alpha : \Omega^k \to \mathbf{R}$ define α^i to be the projection of α on the ith coordinate. Thus, for $a \in \Omega$

$$\alpha^i(a) = \sum_{\{z \in \Omega^k \mid z_i = a\}} \alpha(z).$$

In particular for $\alpha : \bar{X} \times \bar{Y} \to \mathbf{R}$, define $\Omega = X \times Y$, and think of α as a measure (or function) $\alpha : \Omega^k \to \mathbf{R}$. The projection α^i is now a measure (or function) $\alpha^i : X \times Y \to \mathbf{R}$.

In particular we will be interested in the projections $\bar{\pi}^i$.

For a probability measure $\gamma : X \times Y \to \mathbf{R}^+$, we will be interested, from time to time, in the γ-probability of an element $x \in X$. We denote this probability by $\gamma(x)$; thus

$$\gamma(x) = \sum_{y \in Y} \gamma(x,y).$$

For simplicity we use the same notation $\gamma(y)$ for the γ-probability of an element $y \in Y$. The difference will be that we use the letters x, x_i, a, a_i, for elements of X, and the letters y, y_i, b, b_i, for elements of Y. It will always be clear if the element is an element of X or an element of Y.

When X, Y, U, V, Q are fixed, we will be interested in the value of the game, $w(\gamma)$, as a function of the measure $\gamma : X \times Y \to \mathbf{R}^+$. First notice that w is a continuous function. Also, if for γ_1, γ_2, the L_1 distance satisfies $\| \gamma_1 - \gamma_2 \|_1 \le \epsilon$ then $|w(\gamma_1) - w(\gamma_2)| \le \epsilon$. This is true since every protocol for one of the measures can be viewed as a protocol for the other one, with value different by at most ϵ. Thus w has a Lipschitz constant of 1.

If $\gamma = p\gamma_1 + (1-p)\gamma_2$, for $0 \le p \le 1$, then

$$w(\gamma) \le pw(\gamma_1) + (1-p)w(\gamma_2).$$

This is true because every protocol for γ can be viewed as a protocol for γ_1 and γ_2. Thus, the function w is concave. If $\gamma = \sum_{i=1}^{m} p_i \gamma_i$, where $\forall i : 0 \leq p_i \leq 1$, and $\sum_{i=1}^{m} p_i = 1$, then

$$w(\gamma) \leq \sum_{i=1}^{m} p_i w(\gamma_i).$$

For a game G, it is sometimes convenient to consider also probabilistic protocols. In a probabilistic protocol, the "answers" of the players can depend also on a random string. Thus the "answers" are $u(x,h), v(y,h)$ (rather than $u(x), v(y)$), where h is a random string. The value of the protocol will be the probability, over the inputs (x,y), and over the random strings that $Q(x,y,u(x,h),v(y,h)) = 1$.

However, since a probabilistic protocol can be viewed as a convex combination of deterministic ones, the value of any probabilistic protocol can be achieved by a deterministic one.

Remark. In this paper the logarithm function log is always taken base 2. The natural logarithm is denoted by ln.

1.5. Organization of the paper. The paper is organized as follows: In section 2, we show how Theorem 1.2 leads to the proof of Theorem 1.1. In section 3, we review the definition, and the basic properties of the informational divergence, a basic tool of information theory. This tool is needed for the proof of Theorem 1.2. Theorem 1.2 is proved in section 4. The proof of the main lemma is deferred to section 5, and the proof of another lemma is deferred to section 6. In section 7 several generalizations of Theorem 1.1 are shortly discussed.

We remark that two "shortcuts" can be done in the paper. First, in the special case where the measure μ is a product measure (i.e., $\mu = \mu_X \times \mu_Y$), the entire argument is much simpler, and the proof follows simply from section 2 and the beginning of section 4 (plus simpler versions of several lemmas in section 3, using entropy instead of informational divergence). Also, section 6 is not really needed for the proof of the parallel repetition conjecture, but it is needed to improve the constant in the exponent to $\log(s(G))$. A simpler proof which does not use section 6 can be given, but the constant in the exponent in that proof is much worse.

2. Proof of the parallel repetition theorem. In this section, we show how Theorem 1.2 leads to the proof of Theorem 1.1.

Theorem 1.1 claims an upper bound for the value $w(\bar{G})$. We will prove even more: we will upper bound the value $w(\bar{G}_A)$, of the game $\bar{G}_A$, for any large product set $A = A_X \times A_Y \subset \bar{X} \times \bar{Y}$.

We will first give some intuition: The proof uses a simple induction on the dimension k. The idea is to assume by Theorem 1.2 w.l.o.g. an upper bound for $w(\bar{G}^1_A)$. Given a protocol for $\bar{G}_A$, we partition A into product subsets, according to the behavior of the protocol on the first coordinate. The size of this partition is not too big, and therefore, the average size of a subset in the partition is not too small. In many of these subsets the protocol fails to satisfy $\bar{Q}$, because it fails to satisfy $\bar{Q}^1$. We can disregard these subsets. In every other subset, the predicate $\bar{Q}$ can be thought of as a $k-1$ dimensional predicate, and we can use induction to upper bound the size of the set of points, which satisfy this predicate. By this argument, we will get a recursive bound for $w(G_A^{\otimes k})$, as a function of the dimension k, and the size $\bar{\mu}(A)$.

For the game G, define $C_G(k,r)$ to be the maximum, taken over A, of the value $w(G_A^{\otimes k})$, of a game $G_A^{\otimes k} = \bar{G}_A = \bar{G}_{\bar{\pi}}$, where $A = A_X \times A_Y$ is a product set (with

$A_X \subset \bar{X}, A_Y \subset \bar{Y}$), with

$$-\log \bar{\mu}(A) \leq r$$

(i.e., $\bar{\mu}(A) \geq 2^{-r}$). Here, k is the dimension and $r \geq 0$ is real. For $k = 0$, it will be convenient to define $C_G(0, r) = 1$. We will prove an upper bound for $C_G(k, r)$ as a function of $w(G)$. The theorem will follow by taking $r = 0$.

Recall that X, Y, U, V, Q, μ refer to the game G from Theorem 1.1 and that $s = s(G)$. Assume for simplicity that $0 < w(G) < 1$ (otherwise the game is trivial). Take $0 < \Delta < 1$, (Δ will be determined later on). For $r \leq \Delta k$, take $A = A_X \times A_Y \subset \bar{X} \times \bar{Y}$, with $-\log \bar{\mu}(A) \leq r$, which achieves $C_G(k, r)$. Thus,

$$w(\bar{G}_{\bar{\pi}}) = C_G(k, r).$$

By Theorem 1.2, there exists i with

$$w(\bar{G}^i_{\bar{\pi}}) = w_i(\bar{\pi}) \leq W_2(w(G)) + c_0 \Delta^{1/16}$$

where W_2, c_0 are taken from Theorem 1.2. Without loss of generality assume that $i = 1$. Denote

$$\hat{w} = W_2(w(G)) + c_0 \Delta^{1/16}.$$

We will later on assume that Δ is such that

$$0 < \hat{w} < 1$$

and

$$2^{-\frac{1}{2}} < \hat{w}^{\frac{1}{2(\log(s)+\Delta)}} < 1$$

(at this point, the reader can ignore these two assumptions).

Let $u : \bar{X} \to \bar{U}$, $v : \bar{Y} \to \bar{V}$ be a protocol for $\bar{G}_{\bar{\pi}}$, achieving the value $w(\bar{G}_{\bar{\pi}})$. The pair $(x_1, u_1(x))$ is a function of $x \in \bar{X}$. Partition A_X according to $(x_1, u_1(x))$. Formally, $\forall x' \in X, u' \in U$ define

$$A_X(x', u') = \{x \in A_X \mid x_1 = x', u_1(x) = u'\}.$$

Then the family $\{A_X(x', u')\}$ is a partition of A_X. In the same way, define

$$A_Y(y', v') = \{y \in A_Y \mid y_1 = y', v_1(y) = v'\}.$$

Then the family $\{A_Y(y', v')\}$ is a partition of A_Y. For simplicity, denote in all the following $Z = X \times Y \times U \times V$ and $z = (x', y', u', v') \in Z$. For all $z = (x', y', u', v') \in Z$ denote

$$A(z) = A_X(x', u') \times A_Y(y', v').$$

Then the family $\{A(z)\}$ is a partition of A, and we have

$$A = \bigcup_{z \in Z} A(z).$$

For all $z \in Z$, define

$$B(z) = \left\{(x, y) \in A(z) \mid \bar{Q}(x, y, u(x), v(y)) = 1\right\},$$

i.e., $B(z)$ is just the set of elements of $A(z)$ satisfying $\bar{Q}$. Notice that for $z \notin Q$ (i.e., z such that $Q(z) = 0$) we have $(x, y) \in A(z) \Rightarrow Q(x_1, y_1, u_1(x), v_1(y)) = Q(z) = 0$. Therefore, $z \notin Q$ implies $B(z) = \emptyset$. On the other hand, for $z \in Q$ we have that $(x, y) \in A(z) \Rightarrow Q(x_1, y_1, u_1(x), v_1(y)) = Q(z) = 1$. Therefore, for $z \in Q$ and $(x, y) \in A(z)$,

$$\bar{Q}(x, y, u(x), v(y)) = \prod_{i=2}^{k} Q(x_i, y_i, u_i(x), v_i(y)).$$

Thus in this case, $B(z)$ is a set of elements satisfying a $k-1$-dimensional predicate. This fact enables us to use induction.

For all z define

$$\alpha(z) = \bar{\pi}(A(z)), \quad \beta(z) = \frac{\bar{\pi}(B(z))}{\bar{\pi}(A(z))}.$$

Then we have the following.

CLAIM 2.1.

$$\sum_{z \in Q} \alpha(z) \leq \hat{w}.$$

Proof. $u_1 : \bar{X} \to U$, $v_1 : \bar{Y} \to V$ can be viewed as a protocol for $\bar{G}^1_{\bar{\pi}}$. The value of this protocol is clearly

$$\sum_{z \in Q} \bar{\pi}(A(z)) = \sum_{z \in Q} \alpha(z),$$

but this value is at most $w(\bar{G}^1_{\bar{\pi}}) \leq \hat{w}$. □

CLAIM 2.2. *For all $z = (x', y', u', v') \in Q$, with $\alpha(z) > 0$,*

$$\beta(z) \leq C_G(k-1, r - \log[\alpha(z)/\mu(x', y')]).$$

Proof. First notice that if $\alpha(z) > 0$ then also $\mu(x', y') > 0$, thus the logarithm is well defined.

For $k = 1$, the claim is immediate. Assume that $k > 1$. Ignoring the first coordinate, which is fixed, $A(z)$ can be viewed as a set of dimension $k-1$. Formally, define $A'(z) \subset X^{k-1} \times Y^{k-1}$ by

$$A'(z) = \left\{ (\tilde{x}, \tilde{y}) \in X^{k-1} \times Y^{k-1} \;\middle|\; ((x', \tilde{x}), (y', \tilde{y})) \in A(z) \right\}$$

where $(x', \tilde{x})$ denotes $x \in X^k$, with $x_1 = x'$, and $(x_2, \ldots, x_k) = \tilde{x}$ (and, similarly, $(y', \tilde{y})$).

Since for $(x, y) \in A(z)$: $x_1 = x'$, and $y_1 = y'$, we have by definition

$$\mu^{\otimes k}(A(z)) = \mu(x', y') \mu^{\otimes k-1}(A'(z)).$$

In the same way, define

$$B'(z) = \left\{ (\tilde{x}, \tilde{y}) \in X^{k-1} \times Y^{k-1} \;\middle|\; ((x', \tilde{x}), (y', \tilde{y})) \in B(z) \right\}.$$

Then we have

$$\mu^{\otimes k}(B(z)) = \mu(x', y') \mu^{\otimes k-1}(B'(z)).$$

The last $k-1$ coordinates of $u\colon \bar{X} \to \bar{U}, v\colon \bar{Y} \to \bar{V}$ can be viewed as a protocol for the game $G_{A'(z)}^{\otimes k-1}$. Since $z \in Q$, this protocol satisfies $Q^{\otimes k-1}$ at the set of elements $B'(z)$. Therefore, the value of this protocol is

$$\frac{\mu^{\otimes k-1}(B'(z))}{\mu^{\otimes k-1}(A'(z))} = \frac{\bar{\mu}(B(z))}{\bar{\mu}(A(z))} = \frac{\bar{\mu}(B(z))/\bar{\mu}(A)}{\bar{\mu}(A(z))/\bar{\mu}(A)} = \frac{\bar{\pi}(B(z))}{\bar{\pi}(A(z))} = \beta(z),$$

so by the definition of C_G we have

$$\begin{aligned}\beta(z) &\le C_G(k-1, -\log \mu^{\otimes k-1}(A'(z)) = C_G(k-1, -\log[\bar{\mu}(A(z)/\mu(x',y')]) \\ &= C_G(k-1, -\log[\bar{\mu}(A)\alpha(z)/\mu(x',y')]) \le C_G(k-1, r - \log\alpha(z) + \log\mu(x',y'))\end{aligned}$$

(recall that by definition $C_G(k',r')$ is monotone in r'). Notice that since $\mu^{\otimes k-1}(A'(z)) \le 1$, $r - \log\alpha(z) + \log\mu(x',y') \ge 0$, thus C_G is defined. ∎

Claim 2.3.

$$C_G(k,r) = \sum_{z\in Q} \alpha(z)\beta(z).$$

Proof. The protocol u, v satisfies $\bar{Q}$ at the set of elements $\bigcup_{z\in Z} B(z)$. Therefore,

$$C_G(k,r) = w(\bar{G}_{\bar{\pi}}) = \bar{\pi}\left(\bigcup_{z\in Z} B(z)\right) = \sum_{z\in Z} \bar{\pi}(B(z)) = \sum_{z\in Z} \bar{\pi}(A(z))\frac{\bar{\pi}(B(z))}{\bar{\pi}(A(z))} = \sum_{z\in Z} \alpha(z)\beta(z)$$

but $z \notin Q$ implies $\beta(z) = 0$. Thus,

$$C_G(k,r) = \sum_{z\in Q} \alpha(z)\beta(z). \qquad ∎$$

Denote

$$T = \{z \in Q \mid \alpha(z) > 0\}.$$

From Claims 2.2 and 2.3 we have the recursive inequality

$$C_G(k,r) \le \sum_{z\in T} \alpha(z) C_G(k-1, r - \log[\alpha(z)/\mu(x',y')]) \tag{1}$$

where, by Claim 2.1,

$$\sum_{z\in T} \alpha(z) \le \hat{w}.$$

We will now assume that Δ is such that

$$0 < \hat{w} < 1$$

and

$$2^{-1/2} < \hat{w}^{1/(2(\log(s)+\Delta))} < 1.$$

We will prove by induction on k the following inequality.

CLAIM 2.4.

$$C_G(k,r) \leq \left(\hat{w}^{1/(2(\log(s)+\Delta))}\right)^{\Delta k - r}.$$

Proof. For $k = 0$, the claim is trivial, (since $0 < \hat{w} < 1$, and $r \geq 0$).
For $k \geq 1$ assume the inequality for $k-1$, and substitute in inequality (1) to get

$$\begin{aligned} C_G(k,r) &\leq \sum_{z \in T} \alpha(z) \left(\hat{w}^{1/(2(\log(s)+\Delta))}\right)^{\Delta(k-1)-r+\log[\alpha(z)/\mu(x',y')]} \\ &= \left(\hat{w}^{1/(2(\log(s)+\Delta))}\right)^{\Delta k - r} \sum_{z \in T} \alpha(z) \left(\hat{w}^{1/(2(\log(s)+\Delta))}\right)^{-\Delta+\log[\alpha(z)/\mu(x',y')]}. \end{aligned}$$

Hence, it will be enough to prove

$$\sum_{z \in T} \alpha(z) \left(\hat{w}^{1/(2(\log(s)+\Delta))}\right)^{-\Delta-\log(s)+\log[s\alpha(z)/\mu(x',y')]} \leq 1.$$

Define

$$t(z) = \begin{cases} 0 & \text{for } z \notin T, \\ \dfrac{s\alpha(z)}{\mu(x',y')} & \text{for } z \in T. \end{cases}$$

Then the inequality is equivalent to

$$\sum_{z \in T} \left(\frac{\mu(x',y')}{s}\right) t(z) \left(\hat{w}^{1/(2(\log(s)+\Delta))}\right)^{\log t(z)} \leq \hat{w}^{1/2}.$$

Define $p(z) = \mu(x',y')/s$, and $f : \mathbf{R}^+ - \{0\} \to \mathbf{R}$ by

$$f(t) = tc^{\log t} = t^{1+\log c}$$

where

$$c = \hat{w}^{1/(2(\log(s)+\Delta))}.$$

In these notations, we have to prove

$$\sum_{z \in T} p(z) f(t(z)) \leq \hat{w}^{1/2}.$$

We assumed $2^{-1/2} < c < 1$. Therefore, $-\frac{1}{2} < \log c < 0$, thus $f(t) = tc^{\log t} = t^{1+\log c}$ is convex. Notice that

$$\sum_{z \in Z} p(z) = \sum_{U \times V} \sum_{X \times Y} \frac{\mu(x',y')}{s} = \frac{1}{s} \sum_{U \times V} \sum_{X \times Y} \mu(x',y') = \frac{1}{s} \sum_{U \times V} 1 = \frac{1}{s} s = 1.$$

Therefore, we can use Jensen's inequality to conclude

$$\sum_{z \in T} p(z) f(t(z)) = \sum_{z \in T} p(z) t(z)^{1+\log c} = \sum_{z \in Z} p(z) t(z)^{1+\log c} \leq \left(\sum_{z \in Z} p(z) t(z)\right)^{1+\log c},$$

but

$$\sum_{z\in Z} p(z)t(z) = \sum_{z\in T} p(z)t(z) = \sum_{z\in T} \frac{\mu(x',y')}{s}\frac{s\alpha(z)}{\mu(x',y')} = \sum_{z\in T}\alpha(z) \le \hat{w}$$

and since the function $t^{1+\log c}$ is monotone in t, we have

$$\sum_{z\in T} p(z)f(t(z)) \le \left(\sum_{z\in Z} p(z)t(z)\right)^{1+\log c} \le \hat{w}^{1+\log c} \le \hat{w}^{1-1/2} = \hat{w}^{1/2}$$

(where the third inequality uses the assumption $\log c > -1/2$). □

By Claim 2.4 and by $\Delta < 2\log_2(s)$ (which follows from the assumptions: $\Delta < 1$, and $s \ge 2$), we can now conclude

$$w(G^{\otimes k}) = C_G(k,0) \le \left(\hat{w}^{1/(2(\log(s)+\Delta))}\right)^{\Delta k} \le \left(\hat{w}^{1/(4\log(s))}\right)^{\Delta k} = \left(\hat{w}^{(1/4)\Delta}\right)^{k/\log(s)}$$

where $\hat{w} = W_2(w(G)) + c_0\Delta^{1/16}$, and $0 < \Delta < 1$ satisfies $0 < \hat{w} < 1$, and

$$2^{-1/2} < \hat{w}^{1/(2(\log(s)+\Delta))} < 1.$$

Since $0 < W_2(w(G)) < 1$, and since $\hat{w}$ is monotone in Δ, the conditions are satisfiable. Just start from $\Delta = 0$ and increase Δ until the conditions hold.

Thus Theorem 1.1 follows. We will take in Theorem 1.1

$$W(w(G)) = \inf\left(\hat{w}^{(1/4)\Delta}\right)$$

where the infimum is taken over all $0 < \Delta < 1$, which satisfy the conditions.

3. Informational divergence. In this section, we define the informational divergence, a basic tool of information theory, and review some of its basic properties that will be used in the paper. The reader can find excellent treatments of the subject in [29, 17].

Given a finite probability space Ω, with two probability measures ϑ, ψ, the divergence of ϑ with respect to ψ is defined by

$$\mathbf{D}(\vartheta \parallel \psi) = \sum_{z\in\Omega} \vartheta(z)\log\frac{\vartheta(z)}{\psi(z)}.$$

In this definition, $0\log\frac{0}{0}$ is defined to be 0, and for $z > 0$, $z\log\frac{z}{0}$ is defined to be ∞. Thus $\mathbf{D}(\vartheta \parallel \psi) < \infty$ if and only if ϑ is absolutely continuous with respect to ψ (i.e., $\psi(z) = 0$ implies $\vartheta(z) = 0$).

In the special case where ψ is the uniform distribution, we have

$$\mathbf{D}(\vartheta \parallel \psi) = \log_2|\Omega| - \mathbf{H}(\vartheta),$$

where $\mathbf{H}(\vartheta)$ is the standard entropy of ϑ. More generally, the divergence $\mathbf{D}(\vartheta \parallel \psi)$ can be thought of as the entropy of the measure ϑ, relative to the measure ψ, as opposed to the standard entropy of ϑ, which is taken relative to the uniform distribution.

$\mathbf{D}(\vartheta \parallel \psi)$ has many names and notations throughout the literature. In [29] it is also called "relative entropy," and denoted by $\mathbf{H}_{\vartheta\parallel\psi}(\mathcal{Q})$, where $\mathcal{Q}$ is the partition of Ω

into single points. This is a special case of the following definition: For a measurement f on Ω, with a finite alphabet A, let $\mathcal{Q}$ be the induced partition $\{f^{-1}(a)\}_{a\in A}$. Let ϑ_f, ψ_f be the corresponding probability mass functions, i.e., for $a \in A$

$$\vartheta_f(a) = \vartheta(\{z \in \Omega \mid f(z) = a\}), \quad \psi_f(a) = \psi(\{z \in \Omega \mid f(z) = a\}).$$

The relative entropy of f, with measure ϑ, with respect to the measure ψ, is defined by

$$\mathbf{H}_{\vartheta\|\psi}(f) = \mathbf{H}_{\vartheta\|\psi}(\mathcal{Q}) = \sum_{a\in A} \vartheta_f(a) \log \frac{\vartheta_f(a)}{\psi_f(a)}.$$

In this paper, we prefer to use the notation $\mathbf{D}\left(\vartheta \parallel \psi\right)$.

The following lemma, known as the divergence inequality, is probably the most basic property of the informational divergence.

LEMMA 3.1. *For all ϑ, ψ, we have*

$$\mathbf{D}\left(\vartheta \parallel \psi\right) \geq 0.$$

Proof. See [29, Chapter 2, Theorem 2.3.1]. □

For measures $\bar{\vartheta}, \bar{\psi}$ on $\otimes_{i=1}^{k}\Omega_i$, recall that $\bar{\vartheta}^i, \bar{\psi}^i$ are the projections of $\bar{\vartheta}, \bar{\psi}$ on the ith coordinate.

LEMMA 3.2. *For measures $\bar{\vartheta}, \bar{\psi}$ on $\Omega_1 \times \Omega_2$, such that $\bar{\psi} = \bar{\psi}^1 \times \bar{\psi}^2$,*

$$\mathbf{D}\left(\bar{\vartheta} \parallel \bar{\psi}\right) \geq \mathbf{D}\left(\bar{\vartheta}^1 \parallel \bar{\psi}^1\right) + \mathbf{D}\left(\bar{\vartheta}^2 \parallel \bar{\psi}^2\right).$$

Proof. See [29, Chapter 2, Lemma 2.5.3]. The lemma is stated there as: $M_{XY} = M_X \times M_Y$ implies

$$\mathbf{H}_{P\|M}(X, Y) \geq \mathbf{H}_{P\|M}(X) + \mathbf{H}_{P\|M}(Y). \qquad \square$$

The next lemma can be viewed as a generalization of the well-known entropy inequality

$$\mathbf{H}(z_1, \ldots, z_k) \leq \mathbf{H}(z_1) + \cdots + \mathbf{H}(z_k)$$

(for any random variable z), and as a generalization of the previous lemma.

LEMMA 3.3. *For measures $\bar{\vartheta}, \bar{\psi}$ on Ω^k, such that $\bar{\psi} = \otimes_{i=1}^{k}\bar{\psi}^i$*

$$\mathbf{D}\left(\bar{\vartheta} \parallel \bar{\psi}\right) \geq \sum_{i=1}^{k} \mathbf{D}\left(\bar{\vartheta}^i \parallel \bar{\psi}^i\right).$$

Proof. The proof is immediate from Lemma 3.2. □

For a function $\alpha : \Omega \to \mathbf{R}$, denote by $\| \alpha \|_1$ the standard L_1 norm of α, i.e.,

$$\| \alpha \|_1 = \| \alpha(z) \|_1 = \sum_{z\in\Omega} |\alpha(z)|.$$

If α is a probability measure then $\| \alpha \|_1 = 1$. In this paper we use $\| \vartheta - \psi \|_1$ as a distance function between measures (or functions). It is not true that the divergence $\mathbf{D}\left(\vartheta \parallel \psi\right)$ is a distance function. However, it is true that if $\mathbf{D}\left(\vartheta \parallel \psi\right)$ is small then the L_1 distance between ϑ and ψ is also small.

LEMMA 3.4. *For all ϑ, ψ, we have*

$$(2\ln 2)\mathbf{D}(\vartheta \parallel \psi) \geq (\parallel \vartheta - \psi \parallel_1)^2.$$

Proof. See [17, Chapter 3, Exercise 17] and the references therein. □

The next lemma computes the value of $\mathbf{D}(\vartheta \parallel \psi)$, in the special case $\vartheta = \psi_A$, where $A \subset \Omega$.

LEMMA 3.5. *For a measure ψ, and a set $A \subset \Omega$, we have*

$$\mathbf{D}(\psi_A \parallel \psi) = -\log \psi(A).$$

Proof.

$$\mathbf{D}(\psi_A \parallel \psi) = \sum_{z\in\Omega} \psi_A(z) \log \frac{\psi_A(z)}{\psi(z)} = \sum_{z\in A} \psi_A(z) \log \frac{\psi_A(z)}{\psi(z)}$$

$$= \sum_{z\in A} \psi_A(z) \log \frac{\psi(z)/\psi(A)}{\psi(z)} = \sum_{z\in A} \psi_A(z)(-\log \psi(A)) = -\log \psi(A). \quad \square$$

For a probability measure $\alpha : \Omega \to \mathbf{R}^+$, where $\Omega = X \times Y$, define $\alpha(a,\cdot) : Y \to \mathbf{R}^+$ to be the probability measure on Y, derived from α by fixing $x = a$. Thus, $\alpha(a,\cdot)$ is the following probability measure: for all $y \in Y$,

$$\alpha(a,\cdot)(y) = \frac{\alpha(a,y)}{\alpha(a)}$$

where $\alpha(a) = \sum_{y\in Y} \alpha(a,y)$. This definition makes sense only if $\alpha(a) > 0$. Otherwise, define $\alpha(a,\cdot)$ to be identically 0.

In the same way, define the measure $\alpha(\cdot,y) : X \to \mathbf{R}^+$.

For the measures $\vartheta, \psi : X \times Y \to \mathbf{R}^+$, we will be interested in the values of $\mathbf{D}(\vartheta(x,\cdot) \parallel \psi(x,\cdot))$, and $\mathbf{D}(\vartheta(\cdot,y) \parallel \psi(\cdot,y))$. Define

$$\mathbf{V_X}(\vartheta \parallel \psi) = \sum_{x\in X} \vartheta(x)\mathbf{D}(\vartheta(x,\cdot) \parallel \psi(x,\cdot)),$$

and

$$\mathbf{V_Y}(\vartheta \parallel \psi) = \sum_{y\in Y} \vartheta(y)\mathbf{D}(\vartheta(\cdot,y) \parallel \psi(\cdot,y)).$$

These are denoted in [29] by $\mathbf{H}_{\vartheta\parallel\psi}(Y|X)$, and $\mathbf{H}_{\vartheta\parallel\psi}(X|Y)$. In addition, define

$$\mathbf{V}(\vartheta \parallel \psi) = \frac{1}{2}[\mathbf{V_X}(\vartheta \parallel \psi) + \mathbf{V_Y}(\vartheta \parallel \psi)].$$

The notion $\mathbf{V}(\vartheta \parallel \psi)$ is central in the rest of the paper. In particular, we will be interested in cases were $\mathbf{V}(\vartheta \parallel \psi)$ is small. We saw before that if $\mathbf{D}(\vartheta \parallel \psi)$ is small then $\parallel \vartheta - \psi \parallel_1$ is also small. Is the same true for $\mathbf{V}(\vartheta \parallel \psi)$? Taking $X = Y = \{0,1\}$ and

$$\psi = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix}, \quad \vartheta = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

we have $\mathbf{V}(\vartheta \parallel \psi) = 0$, but still $\parallel \vartheta - \psi \parallel_1 = \frac{1}{2}$. Hence, in general it is not the case.

A measure $\psi : X \times Y \to \mathbf{R}^+$ is called irreducible, if there are no nontrivial partitions $X = X_1 \cup X_2$, $Y = Y_1 \cup Y_2$ such that $\psi(X_1 \times Y_2) = \psi(X_2 \times Y_1) = 0$. Every measure $\psi : X \times Y \to \mathbf{R}^+$ decomposes into its irreducible components. In general, it is true that if $\mathbf{V}(\vartheta \parallel \psi) = 0$ then ϑ has the same components as ψ and behaves like ψ on each one of them, but the ϑ-probability of each component can be different from the ψ-probability.

For irreducible ψ it can be shown that if $\mathbf{V}(\vartheta \parallel \psi) = 0$ then $\vartheta = \psi$, and that if ψ is fixed $\mathbf{V}(\vartheta \parallel \psi) \to 0$ implies $\vartheta \to \psi$. However, this convergence is not uniform. For example, we can take $X = Y = \{0, 1\}$, and

$$\psi = \begin{pmatrix} \frac{1}{2} & \epsilon \\ 0 & \frac{1}{2} - \epsilon \end{pmatrix}, \quad \vartheta = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}.$$

In this case ψ is irreducible, and $\mathbf{V}(\vartheta \parallel \psi) = O(\epsilon)$, but still $\| \vartheta - \psi \|_1 = \frac{1}{2}$.

Therefore, in this paper we will use a different characterization of measures ϑ, with small $\mathbf{V}(\vartheta \parallel \psi)$. This characterization, proved in Lemma 6.6, will intuitively say that in this case there are measures ϑ', ψ', such that ϑ' is very close to ϑ, and ψ' is very close to ψ, and such that ϑ', ψ' have the same irreducible components, and behave the same on each one of them (but not necessarily give the same mass to each component).

We remark that this characterization is not necessary to prove the parallel repetition conjecture. It is done here only to improve the constants.

4. Proof of the main theorem. In this section we give the proof of Theorem 1.2. The proofs of two important lemmas are deferred to sections 5 and 6. Given X, Y, U, V, Q, k, define for any probability measure $\bar{\alpha} : \bar{X} \times \bar{Y} \to \mathbf{R}^+$, the following games:

1. The game $G_{\bar{\alpha}^i}$, consisting of X, Y, U, V, Q, with the measure $\bar{\alpha}^i$.
   Denote the value of this game by $w(\bar{\alpha}^i)$.
2. The game $\bar{G}_{\bar{\alpha}}$, consisting of $\bar{X}, \bar{Y}, \bar{U}, \bar{V}, \bar{Q}$, with the measure $\bar{\alpha}$.
   Denote the value of this game by $\bar{w}(\bar{\alpha})$.
3. The game $\bar{G}^i_{\bar{\alpha}}$, consisting of $\bar{X}, \bar{Y}, U, V$, with the predicate $\bar{Q}^i$, defined by

$$\forall (x, y, u, v) \in \bar{X} \times \bar{Y} \times U \times V \quad : \quad \bar{Q}^i(x, y, u, v) = Q(x_i, y_i, u, v)$$

   and with the measure $\bar{\alpha}$. Denote the value of this game by $w_i(\bar{\alpha})$.

Recall that for any probability measure $\gamma : X \times Y \to \mathbf{R}^+$, we denote by $w(\gamma)$ the value of the game G_γ , consisting of X, Y, U, V, Q, γ.

Theorem 1.2 claims that if $A = A_X \times A_Y$ is large then $w_i(\bar{\pi})$ cannot be too large for all coordinates i. If A is large it is easy to show that for many coordinates i, $\bar{\pi}^i$ is very close to μ, and, therefore, $w(\bar{\pi}^i)$ is very close to $w(\mu)$ and is not too large. Hence, it could be enough to show that for some coordinates i, $w_i(\bar{\pi})$ is upper bounded by some "well behaved" function of $w(\bar{\pi}^i)$.

For any $\bar{\alpha} : \bar{X} \times \bar{Y} \to \mathbf{R}^+$, any protocol for $G_{\bar{\alpha}^i}$ defines also a corresponding protocol, with the same value, for $\bar{G}^i_{\bar{\alpha}}$. This is true because if the distribution of (x, y) is $\bar{\alpha}$ then the distribution of (x_i, y_i) is $\bar{\alpha}^i$. Thus, given an input (x, y), for the game $\bar{G}^i_{\bar{\alpha}}$, (x_i, y_i) can be used as an input for the protocol for $G_{\bar{\alpha}^i}$. The output of this protocol, can be viewed as an output for the game $\bar{G}^i_{\bar{\alpha}}$. Therefore, we always have

$$w(\bar{\alpha}^i) \leq w_i(\bar{\alpha}).$$

The other direction is false, as the the other coordinates can give a lot of information on (x_i, y_i). There is an important special case, however, in which $w(\bar{\alpha}^i) = w_i(\bar{\alpha})$.

LEMMA 4.1. *If there exist* $\alpha_1 : X \times Y \to \mathbf{R}$, $\alpha_2 : \bar{X} \to \mathbf{R}$, $\alpha_3 : \bar{Y} \to \mathbf{R}$ *such that for all* $(x, y) \in \bar{X} \times \bar{Y}$

$$\bar{\alpha}(x, y) = \alpha_1(x_i, y_i)\alpha_2(x)\alpha_3(y),$$

then

$$w(\bar{\alpha}^i) = w_i(\bar{\alpha}).$$

Proof. We will show that in this case a protocol for $\bar{G}^i_{\bar{\alpha}}$ defines a corresponding protocol, with the same value for $G_{\bar{\alpha}^i}$. This will be true because in this special case a pair (x, y), with distribution $\bar{\alpha}$, can simply be created by the two players from a pair (a_i, b_i), with distribution $\bar{\alpha}^i$.

First notice that $\alpha_1, \alpha_2, \alpha_3$ are not unique, as we can multiply α_1 and divide α_2, by the same function $f(x_i)$, as long as for all $a \in X : f(a) \neq 0$. In the same way, we can multiply α_1 and divide α_3, by the same function $g(y_i)$, as long as for all $b \in Y : g(b) \neq 0$. Therefore, we can assume w.l.o.g. that for all a_i, b_i:

$$\sum_{\{x \in \bar{X} \mid x_i = a_i\}} \alpha_2(x), \quad \text{and} \quad \sum_{\{y \in \bar{Y} \mid y_i = b_i\}} \alpha_3(y)$$

are always 0 or 1. Also, we can assume w.l.o.g. that if one of them is 0 then $\alpha_1(a_i, b_i)$ is also 0. Therefore, in this case

$$\begin{aligned}\bar{\alpha}^i(a_i, b_i) &= \sum_{\{(x,y) \in \bar{X} \times \bar{Y} \mid (x_i, y_i) = (a_i, b_i)\}} \alpha_1(a_i, b_i)\alpha_2(x)\alpha_3(y) \\ &= \alpha_1(a_i, b_i) \left(\sum_{\{x \in \bar{X} \mid x_i = a_i\}} \alpha_2(x) \right) \left(\sum_{\{y \in \bar{Y} \mid y_i = b_i\}} \alpha_3(y) \right) = \alpha_1(a_i, b_i).\end{aligned}$$

Notice that now, given $a_i \in X$, with $\bar{\alpha}^i(a_i) \neq 0$, we have $\sum_{\{x \in \bar{X} \mid x_i = a_i\}} \alpha_2(x) = 1$. Therefore, for all $a_i \in X$, with $\bar{\alpha}^i(a_i) \neq 0$, we can define a probability distribution on $\bar{X}$ by

$$\Pr_{a_i}(x) = \begin{cases} 0 & \text{if } x_i \neq a_i, \\ \alpha_2(x) & \text{if } x_i = a_i. \end{cases}$$

Note that $\Pr_{a_i}(x)$ is exactly the α-probability for x, given that $x_i = a_i$. In the same way define for $b_i \in Y$, with $\bar{\alpha}^i(b_i) \neq 0$,

$$\Pr_{b_i}(y) = \begin{cases} 0 & \text{if } y_i \neq b_i, \\ \alpha_3(y) & \text{if } y_i = b_i. \end{cases}$$

Given (a_i, b_i), with $\bar{\alpha}^i(a_i, b_i) \neq 0$, choose randomly $x \in \bar{X}$ according to $\Pr_{a_i}(x)$ and $y \in \bar{Y}$ according to $\Pr_{b_i}(y)$. If (a_i, b_i) are chosen with probability $\bar{\alpha}^i(a_i, b_i)$ then (x, y) are chosen with probability

$$\bar{\alpha}^i(x_i, y_i)\Pr_{x_i}(x)\Pr_{y_i}(y) = \alpha_1(x_i, y_i)\alpha_2(x)\alpha_3(y) = \bar{\alpha}(x, y).$$

Thus, if (a_i, b_i) is a random variable, with distribution $\bar{\alpha}^i$, then (x, y) is a random variable with distribution $\bar{\alpha}$.

Players I, II can use a protocol for $\bar{G}^i_{\bar{\alpha}}$ to define a probabilistic protocol for $G_{\bar{\alpha}^i}$ in the following way: Given an input pair $(a_i, b_i) \in X \times Y$ for the game $G_{\bar{\alpha}^i}$, player I chooses randomly $x \in \bar{X}$, according to $\Pr_{a_i}(x)$, and player II chooses randomly $y \in \bar{Y}$, according to $\Pr_{b_i}(y)$. Since (a_i, b_i) is a random variable with distribution $\bar{\alpha}^i$, (x, y) is a random variable with distribution $\bar{\alpha}$ and can be used as an input for $\bar{G}^i_{\bar{\alpha}}$.

Players I, II can now use the protocol for $\bar{G}^i_{\bar{\alpha}}$ on the input (x, y), as a protocol for $G_{\bar{\alpha}^i}$ on the input (a_i, b_i). Formally, if $u : \bar{X} \to U$, $v : \bar{Y} \to V$ is the protocol for $\bar{G}^i_{\bar{\alpha}}$, and h is the random string used by the players, the protocol for $G_{\bar{\alpha}^i}$ will be $u'(a_i, h) = u(x)$, $v'(b_i, h) = v(y)$, (where x, y are created from a_i, b_i, h by the previous procedure). Since

$$\bar{Q}^i(x, y, u(x), v(y)) = Q(x_i, y_i, u(x), v(y)) = Q(a_i, b_i, u'(a_i, h), v'(a_i, h)),$$

the probability that $Q(a_i, b_i, u'(a_i, h), v'(a_i, h)) = 1$ equals exactly the probability that $\bar{Q}^i(x, y, u(x), v(y)) = 1$. Since (x, y) is a random variable with distribution $\bar{\alpha}$, this probability is the value of the original protocol for $\bar{G}^i_{\bar{\alpha}}$.

Thus we proved that a protocol for $\bar{G}^i_{\bar{\alpha}}$ defines a probabilistic protocol with the same value for $G_{\bar{\alpha}^i}$. It is well known that since a probabilistic protocol can be viewed as a convex combination of deterministic ones, there must exist a deterministic protocol with the same value. □

Since the conditions of Lemma 4.1 do not hold usually, we cannot expect them to hold for the measure $\bar{\pi}$. The idea will be to represent $\bar{\pi}$ as a convex combination of measures that satisfy the conditions of Lemma 4.1 and then to deduce bounds for the values $w_i(\bar{\pi})$.

We remark that in the simpler case, in which the original measure μ is a product measure, the measure $\bar{\pi}$ does satisfy the conditions of Lemma 4.1. Had we considered only this simpler case, the entire proof would have ended here!

Recall that in all the following $\bar{\pi} = \bar{\mu}_A$, where the set A is taken from Theorem 1.2 and satisfies,

$$-\log \bar{\mu}(A) \leq \Delta k.$$

The following definition is probably the most important notion in this paper. This definition enables the representation of $\bar{\pi}$ as a convex combination of measures with nice properties. A similar idea was used before in [35, 41].

A scheme M of type $\mathcal{M}^l$ consists of:

1. A partition of the set of coordinates $[k] - \{l\}$ into $I \cup J$.
2. Values $a_i \in X$, for all $i \in I$, and $b_j \in Y$, for all $j \in J$.

(Formally, M should be denoted by $M^l_{I,J,a_{i_1},\ldots,a_{i_{|I|}},b_{j_1},\ldots,b_{j_{|J|}}}$; however, for simplicity we will just use the notation M.) We also denote by M the set

$$M = \left\{ (x, y) \in \bar{X} \times \bar{Y} \;\middle|\; \forall i \in I \;:\; x_i = a_i \;,\; \forall j \in J \;:\; y_j = b_j \right\}.$$

We also denote by $\mathcal{M}^l$ the family of all sets M of type $\mathcal{M}^l$ (i.e., for all possible $I, J, a_{i_1}, \ldots, a_{i_{|I|}}, b_{j_1}, \ldots, b_{j_{|J|}}$). Notice that each $\mathcal{M}^l$ is a cover of $\bar{X} \times \bar{Y}$. Each element $(x, y) \in \bar{X} \times \bar{Y}$ is covered 2^{k-1} times by each $\mathcal{M}^l$.

For all l, define the following two probability measures, $\nu_l, \rho_l : \mathcal{M}^l \to \mathbf{R}^+$ by

$$\nu_l(M) = \frac{\bar{\mu}(M)}{2^{k-1}}, \quad \rho_l(M) = \frac{\bar{\pi}(M)}{2^{k-1}} = \frac{\bar{\mu}_A(M)}{2^{k-1}} = \frac{\bar{\mu}(A \cap M)}{\bar{\mu}(A) 2^{k-1}}$$

(ν_l, ρ_l are obviously probability measures, by the above observation that each element (x, y) is covered exactly 2^{k-1} times by $\mathcal{M}^l$).

Recall that for the set M, we have the measures $\bar{\mu}_M : \bar{X} \times \bar{Y} \to \mathbf{R}^+$, $\bar{\pi}_M : \bar{X} \times \bar{Y} \to \mathbf{R}^+$. For M with $\bar{\mu}(M) > 0$ (respectively, $\bar{\pi}(M) > 0$), these measures are probability measures (recall that otherwise they are defined to be identically 0). Recall that $\bar{\mu}_M^i, \bar{\pi}_M^i : X \times Y \to \mathbf{R}^+$ are the projections of these measures on the ith coordinate. For $M \in \mathcal{M}^l$, we will be mainly interested in the projections $\bar{\mu}_M^l, \bar{\pi}_M^l$. Note that since $\bar{\mu} = \mu^{\otimes k}$, the projection $\bar{\mu}_M^l$ is just μ.

The important fact for M is the following.

CLAIM 4.1. *For $M \in \mathcal{M}^l$ (with $\rho_l(M) > 0$),*

$$w_l(\bar{\pi}_M) = w(\bar{\pi}_M^l).$$

Proof. This is a simple application of Lemma 4.1. For $(x, y) \in M$, we have

$$\begin{aligned}\bar{\mu}_M(x,y) = \bar{\mu}(M)^{-1}\bar{\mu}(x,y) \;&=\; \bar{\mu}(M)^{-1}\prod_{i=1}^{k}\mu(x_i,y_i)\\ &= \bar{\mu}(M)^{-1}\mu(x_l,y_l)\prod_{i\in I}\mu(x_i,y_i)\prod_{j\in J}\mu(x_j,y_j)\\ &= \bar{\mu}(M)^{-1}\mu(x_l,y_l)\prod_{i\in I}\mu(a_i,y_i)\prod_{j\in J}\mu(x_j,b_j).\end{aligned}$$

Define $\mu_1 : X \times Y \to \mathbf{R}$, $\mu_2 : \bar{X} \to \mathbf{R}$, $\mu_3 : \bar{Y} \to \mathbf{R}$ by

$$\mu_1(x_l,y_l) = \bar{\mu}(M)^{-1}\mu(x_l,y_l),\;\; \mu_2(x) = \prod_{j\in J}\mu(x_j,b_j),\;\; \mu_3(y) = \prod_{i\in I}\mu(a_i,y_i).$$

Then for $(x, y) \in M$, we have

$$\bar{\mu}_M(x,y) = \mu_1(x_l,y_l)\mu_2(x)\mu_3(y).$$

The event $(x, y) \in M \cap A$ can be written as

$$(x \in A_X \;\cap\; \forall i \in I : x_i = a_i) \bigcap (y \in A_Y \;\cap\; \forall j \in J : y_j = b_j).$$

Therefore, defining $\chi_2 : \bar{X} \to \mathbf{R}$, $\chi_3 : \bar{Y} \to \mathbf{R}$ by

$$\chi_2(x) = \begin{cases} 1 & \text{if } x \in A_X \;\cap\; \forall i \in I \;:\; x_i = a_i,\\ 0 & \text{otherwise,}\end{cases}$$

$$\chi_3(y) = \begin{cases} 1 & \text{if } y \in A_Y \;\cap\; \forall j \in J \;:\; y_j = b_j,\\ 0 & \text{otherwise,}\end{cases}$$

we have $\forall (x, y) \in \bar{X} \times \bar{Y}$

$$\bar{\pi}_M(x,y) = \bar{\mu}_{A\cap M}(x,y) = (\bar{\mu}_M)_A(x,y) = \bar{\mu}_M(A)^{-1}\chi_2(x)\chi_3(y)\bar{\mu}_M(x,y),$$

and since $(x, y) \notin M$ implies $\chi_2(x)\chi_3(y) = 0$, we have

$$\bar{\pi}_M(x,y) = \bar{\mu}_M(A)^{-1}\chi_2(x)\chi_3(y)\mu_1(x_l,y_l)\mu_2(x)\mu_3(y).$$

Define

$$\begin{aligned}\pi_1(x_l,y_l) &= \bar{\mu}_M(A)^{-1}\mu_1(x_l,y_l),\\ \pi_2(x) &= \chi_2(x)\mu_2(x),\\ \pi_3(y) &= \chi_3(y)\mu_3(y).\end{aligned}$$

Then $\bar{\pi}_M(x, y) = \pi_1(x_l, y_l)\pi_2(x)\pi_3(y)$ and the claim follows from Lemma 4.1. ∎

Recall the definitions of the probability measures ν_l, ρ_l. Since every (x,y) is covered the same number of times by $\mathcal{M}^l$, for any l, the measure $\bar{\pi}$ can be written as the convex combination

$$\bar{\pi} = \sum_{M \in \mathcal{M}^l} \rho_l(M) \bar{\pi}_M. \tag{2}$$

We will use the equality (2) in two ways. First note that a protocol for $\bar{G}^l_{\bar{\pi}}$ is also a protocol for each $\bar{G}^l_{\bar{\pi}_M}$. Therefore,

$$w_l(\bar{\pi}) \leq \sum_{M \in \mathcal{M}^l} \rho_l(M) w_l(\bar{\pi}_M) = \mathbf{E}_{\rho_l}\left(w_l(\bar{\pi}_M)\right),$$

(which reflects the concavity of the function w_l–see in the Introduction), and by Claim 4.1 we have

$$w_l(\bar{\pi}) \leq \mathbf{E}_{\rho_l}\left(w\left(\bar{\pi}^l_M\right)\right). \tag{3}$$

Thus, if we can only upper bound the values $w\left(\bar{\pi}^l_M\right)$, of the one-dimensional measures $\bar{\pi}^l_M$, we will have an upper bound for $w_l(\bar{\pi})$.

In order to deduce an upper bound for $\mathbf{E}_{\rho_l}\left(w\left(\bar{\pi}^l_M\right)\right)$, we need some more properties of the family $\left\{\bar{\pi}^l_M\right\}_{M \in \mathcal{M}^l}$: First take the projection of equality (2) to get

$$\bar{\pi}^l = \sum_{M \in \mathcal{M}^l} \rho_l(M) \bar{\pi}^l_M.$$

We also need the following lemma.

LEMMA 4.2 (main). *There exists l_0 such that*

$$\mathbf{E}_{\rho_{l_0}}\left(\mathbf{V}\left(\bar{\pi}^{l_0}_M \,\middle\|\, \mu\right)\right) \leq \frac{2}{k}(-\log \bar{\mu}(A))$$

and

$$\mathbf{D}\left(\bar{\pi}^{l_0} \,\|\, \mu\right) \leq \frac{2}{k}(-\log \bar{\mu}(A)).$$

The proof is given in the next section.

In order to understand the intuition behind Lemma 4.2, one should think of the right-hand side, $\frac{2}{k}(-\log \bar{\mu}(A))$, as a very small ϵ. The lemma just says that there exists l_0 s.t. $\bar{\pi}^{l_0}$ is very close to μ (by Lemma 3.4), and s.t. for an average M, $\mathbf{V}(\bar{\pi}^{l_0}_M \| \mu)$ is very small.

Thus fixing l_0 from the last lemma, and fixing $\epsilon = 2/k(-\log \bar{\mu}(A))$, the family of measures $\{\bar{\pi}^{l_0}_M\}_{M \in \mathcal{M}^{l_0}}$ satisfies

$$\sum_{M \in \mathcal{M}^{l_0}} \rho_{l_0}(M) \mathbf{V}\left(\bar{\pi}^{l_0}_M \,\middle\|\, \mu\right) \leq \epsilon$$

and

$$\mathbf{D}\left(\sum_{M \in \mathcal{M}^{l_0}} \rho_{l_0}(M) \bar{\pi}^{l_0}_M \,\middle\|\, \mu\right) = \mathbf{D}\left(\bar{\pi}^{l_0} \,\|\, \mu\right) \leq \epsilon.$$

In these conditions, the following lemma proves an upper bound for

$$\sum_{M \in \mathcal{M}^{l_0}} \rho_{l_0}(M) w\left(\bar{\pi}_M^{l_0}\right) = \mathbf{E}_{\rho_{l_0}}\left(w\left(\bar{\pi}_M^{l_0}\right)\right),$$

i.e., an upper bound for the value $w\left(\bar{\pi}_M^{l_0}\right)$, for an average M.

LEMMA 4.3. *There exists a global function $W_2 : [0,1] \to [0,1]$, with $z < 1$ implying $W_2(z) < 1$, and a global constant c_1 such that: if $\{\gamma_d\}_{d \in D}$ is a family of probability measures on $X \times Y$, and $\{p_d\}_{d \in D}$ are weights satisfying $\forall d : p_d \geq 0$, and $\sum_{d \in D} p_d = 1$. Assume that for some $0 \leq \epsilon \leq 1$*

$$\sum_D p_d \mathbf{V}(\gamma_d \parallel \mu) \leq \epsilon$$

and

$$\mathbf{D}\left(\sum_D p_d \gamma_d \,\middle\|\, \mu\right) \leq \epsilon.$$

Then,

$$\sum_D p_d w(\gamma_d) \leq W_2(w(\mu)) + c_1 \epsilon^{1/16}.$$

The proof is given in section 6.

Using Lemma 4.3, and inequality (3), and the fact that

$$\epsilon \stackrel{\text{def}}{=} \frac{2}{k}(-\log \bar{\mu}(A)) \leq \frac{2}{k}\Delta k \leq 2\Delta,$$

we can now conclude

$$w_{l_0}(\bar{\pi}) \leq \mathbf{E}_{\rho_{l_0}}\left(w\left(\bar{\pi}_M^{l_0}\right)\right) \leq W_2(w(\mu)) + c_1(2\Delta)^{1/16},$$

which proves Theorem 1.2.

Lemma 4.2, which is the most important lemma in the paper, is proved in the next section. Lemma 4.3 is proved in section 6.

We remark that Lemma 4.3 is not really necessary: for small enough ϵ, and irreducible μ, the fact that $\mathbf{V}(\gamma_d \parallel \mu) \leq \epsilon$ implies that $\parallel \gamma_d - \mu \parallel_1$ is very small. Therefore, it can be proved easily that

$$\mathbf{E}_{\rho_{l_0}}\left(w\left(\bar{\pi}_M^{l_0}\right)\right) \approx w(\mu),$$

which implies the parallel repetitions conjecture for irreducible measures. It turns out that the general case follows easily from the irreducible one.

However, as we mentioned in section 3, only for a very small ϵ it is the case that $\mathbf{V}(\gamma_d \parallel \mu) \leq \epsilon$ implies that $\parallel \gamma_d - \mu \parallel_1$ is small. In fact, such an ϵ needs to be much smaller than the ϵ used here. In particular, since the measure μ is arbitrary, such an ϵ depends on the size of the input set, $|X \times Y|$, as opposed to the ϵ used here, which depends only on $w(\mu)$ and is independent of other parameters of the game. Thus using Lemma 4.3 improves the constants in the exponent (in our analysis) in the parallel repetition theorem.

5. The main lemma. In this section we give the proof of Lemma 4.2. Our proof uses tools, and intuition from [41].

The main idea of the proof is to introduce a new type of scheme. A scheme M of type $\mathcal{M}$ consists of

1. a partition of the set of coordinates $[k]$ into $I \cup J$;
2. values $a_i \in X$, for all $i \in I$, and $b_j \in Y$, for all $j \in J$;

Note that this is the same as before, except that here $I \cup J = [k]$ rather than $[k] - \{l\}$. As before, we also denote by M the set

$$M = \left\{ (x,y) \in \bar{X} \times \bar{Y} \;\middle|\; \forall i \in I \;:\; x_i = a_i \;,\; \forall j \in J \;:\; y_j = b_j \right\}$$

and by $\mathcal{M}$ the family of all the sets M of type $\mathcal{M}$. Notice that each element $(x,y) \in \bar{X} \times \bar{Y}$ is covered 2^k times, by the cover $\mathcal{M}$. As before, define two probability measures $\nu, \rho : \mathcal{M} \to \mathbf{R}^+$ by

$$\nu(M) = \frac{\bar{\mu}(M)}{2^k} \quad , \quad \rho(M) = \frac{\bar{\pi}(M)}{2^k} = \frac{\bar{\mu}_A(M)}{2^k} = \frac{\bar{\mu}(A \cap M)}{\bar{\mu}(A) 2^k}.$$

As before, we have for the set M the measures $\bar{\mu}_M : \bar{X} \times \bar{Y} \to \mathbf{R}^+$, and $\bar{\pi}_M = \bar{\mu}_{M \cap A} : \bar{X} \times \bar{Y} \to \mathbf{R}^+$. Recall that $\bar{\mu}^i_M, \bar{\pi}^i_M$ are the projections of these measures on the ith coordinate.

For $i \in I$, $x_i = a_i$ is fixed. In this case $\bar{\mu}^i_M, \bar{\pi}^i_M$ are concentrated on $\{a_i\} \times Y$ and can also be thought of as measures on Y. In the same way, if $j \in J$ then $\bar{\mu}^j_M, \bar{\pi}^j_M$ can be thought of as measures on X.

Notice also that since $\bar{\mu} = \mu^{\otimes k}$, we have

$$\bar{\mu}_M = \otimes_{i=1}^k \bar{\mu}^i_M.$$

Claim 5.1.

$$\sum_{i=1}^k \mathbf{E}_\rho \left(\mathbf{D} \left(\bar{\pi}^i_M \parallel \bar{\mu}^i_M \right) \right) \leq - \log \bar{\mu}(A).$$

Proof. The proof follows from the basic properties of informational divergence: For any scheme $M \in \mathcal{M}$ with $\rho(M) > 0$, we have by Lemma 3.5,

$$\begin{aligned} \mathbf{D} \left(\bar{\pi}_M \parallel \bar{\mu}_M \right) &= \mathbf{D} \left(\bar{\mu}_{M \cap A} \parallel \bar{\mu}_M \right) = \mathbf{D} \left((\bar{\mu}_M)_A \parallel \bar{\mu}_M \right) \\ &= - \log \bar{\mu}_M(A) = - \log \frac{\bar{\mu}(M \cap A)}{\bar{\mu}(M)} = - \log \frac{\rho(M) \bar{\mu}(A)}{\nu(M)}. \end{aligned}$$

Therefore,

$$\begin{aligned} \mathbf{E}_\rho \left(\mathbf{D} \left(\bar{\pi}_M \parallel \bar{\mu}_M \right) \right) &= - \sum_{M \in \mathcal{M}} \rho(M) \log \frac{\rho(M) \bar{\mu}(A)}{\nu(M)} \\ &= - \sum_{M \in \mathcal{M}} \rho(M) \left[\log \frac{\rho(M)}{\nu(M)} + \log \bar{\mu}(A) \right] \\ &= -\mathbf{D} \left(\rho \parallel \nu \right) - \log \bar{\mu}(A) \;\; \leq \;\; - \log \bar{\mu}(A) \end{aligned}$$

(by Lemma 3.1). But by Lemma 3.3, for all M, with $\rho(M) > 0$,

$$\mathbf{D} \left(\bar{\pi}_M \parallel \bar{\mu}_M \right) \geq \sum_{i=1}^k \mathbf{D} \left(\bar{\pi}^i_M \parallel \bar{\mu}^i_M \right)$$

and we can conclude that

$$\sum_{i=1}^{k} \mathbf{E}_\rho \left(\mathbf{D} \left(\bar{\pi}^i_M \parallel \bar{\mu}^i_M \right) \right) = \mathbf{E}_\rho \left(\sum_{i=1}^{k} \mathbf{D} \left(\bar{\pi}^i_M \parallel \bar{\mu}^i_M \right) \right)$$
$$\leq \mathbf{E}_\rho \left(\mathbf{D} \left(\bar{\pi}_M \parallel \bar{\mu}_M \right) \right) \leq -\log \bar{\mu}(A). \qquad \square$$

Fix l. In what follows we denote by M a scheme of type $\mathcal{M}$, and by M' a scheme of type $\mathcal{M}^l$. Denote by I, J the partition of coordinates, corresponding to the scheme M, and by I', J' the one corresponding to M'. A scheme M agrees with a scheme M' if and only if $I' = I \cap ([k] - \{l\})$, $J' = J \cap ([k] - \{l\})$, and M, M' agree on the values of a_i, b_j, for all $i \in I', j \in J'$. For a scheme M' of type $\mathcal{M}^l$, denote the set of all schemes M of type $\mathcal{M}$, agreeing with M', by $N(M')$. Then the family of sets $\{M\}_{M \in N(M')}$ is a cover of the set M', where each element $(x, y) \in M'$ is covered exactly twice (as we have to choose $l \in I$ or $l \in J$ and then fix the value of a_l or b_l). Therefore, we have

$$\bar{\pi}(M') = \frac{1}{2} \sum_{M \in N(M')} \bar{\pi}(M),$$

and hence,

$$\rho_l(M') = \sum_{M \in N(M')} \rho(M).$$

Assume in all of the following that $\rho_l(M') > 0$. Then $\rho(M)/\rho_l(M')$ is a probability measure on $N(M')$. A scheme M can be randomly chosen, according to the distribution ρ, by first choosing M' according to ρ_l and then choosing $M \in N(M')$ with probability $\rho(M)/\rho_l(M')$.

CLAIM 5.2. *For every l,*

$$\mathbf{V} \left(\bar{\pi}^l_{M'} \parallel \bar{\mu}^l_{M'} \right) = \sum_{M \in N(M')} \frac{\rho(M)}{\rho_l(M')} \mathbf{D} \left(\bar{\pi}^l_M \parallel \bar{\mu}^l_M \right).$$

Proof. The proof follows by looking carefully into the definitions. For $M \in N(M')$

$$\frac{\rho(M)}{\rho_l(M')} = \frac{\bar{\pi}(M)/2^k}{\bar{\pi}(M')/2^{k-1}} = \frac{1}{2} \frac{\bar{\pi}(M)}{\bar{\pi}(M')} = \frac{1}{2} \bar{\pi}_{M'}(M).$$

If for the scheme M, $l \in I$, then $\bar{\pi}_{M'}(M)$ is just the probability to have $x_l = a_l$ in the set M'. By definition this probability is $\bar{\pi}^l_{M'}(a_l)$. Therefore, in this case,

$$\frac{\rho(M)}{\rho_l(M')} = \frac{1}{2} \bar{\pi}^l_{M'}(a_l).$$

Also in this case, $\bar{\pi}_M$ is derived from $\bar{\pi}_{M'}$ by fixing x_l to be a_l. Hence, $\bar{\pi}^l_M = \bar{\pi}^l_{M'}(a_l, \cdot)$. In the same way, we have $\bar{\mu}^l_M = \bar{\mu}^l_{M'}(a_l, \cdot)$. Therefore, for $l \in I$

$$\frac{\rho(M)}{\rho_l(M')} \mathbf{D} \left(\bar{\pi}^l_M \parallel \bar{\mu}^l_M \right) = \frac{1}{2} \bar{\pi}^l_{M'}(a_l) \mathbf{D} \left(\bar{\pi}^l_{M'}(a_l, \cdot) \parallel \bar{\mu}^l_{M'}(a_l, \cdot) \right).$$

In the same way, for $l \in J$

$$\frac{\rho(M)}{\rho_l(M')} \mathbf{D} \left(\bar{\pi}^l_M \parallel \bar{\mu}^l_M \right) = \frac{1}{2} \bar{\pi}^l_{M'}(b_l) \mathbf{D} \left(\bar{\pi}^l_{M'}(\cdot, b_l) \parallel \bar{\mu}^l_{M'}(\cdot, b_l) \right).$$

Partitioning $M \in N(M')$ into schemes with $l \in I$, and schemes with $l \in J$, we have

$$\begin{aligned}
&\sum_{M \in N(M')} \frac{\rho(M)}{\rho_l(M')} \mathbf{D}\left(\bar{\pi}^l_M \parallel \bar{\mu}^l_M\right) \\
&= \sum_{a \in X} \frac{1}{2} \bar{\pi}^l_{M'}(a) \mathbf{D}\left(\bar{\pi}^l_{M'}(a,\cdot) \parallel \bar{\mu}^l_{M'}(a,\cdot)\right) + \sum_{b \in Y} \frac{1}{2} \bar{\pi}^l_{M'}(b) \mathbf{D}\left(\bar{\pi}^l_{M'}(\cdot,b) \parallel \bar{\mu}^l_{M'}(\cdot,b)\right) \\
&= \frac{1}{2} \mathbf{V}_{\mathbf{X}}\left(\bar{\pi}^l_{M'} \parallel \bar{\mu}^l_{M'}\right) + \frac{1}{2} \mathbf{V}_{\mathbf{Y}}\left(\bar{\pi}^l_{M'} \parallel \bar{\mu}^l_{M'}\right) \;=\; \mathbf{V}\left(\bar{\pi}^l_{M'} \parallel \bar{\mu}^l_{M'}\right). \qquad \square
\end{aligned}$$

Claim 5.3.

$$\sum_{l=1}^{k} \mathbf{E}_{\rho_l}\left(\mathbf{V}\left(\bar{\pi}^l_{M'} \parallel \bar{\mu}^l_{M'}\right)\right) \leq -\log \bar{\mu}(A).$$

Proof. By Claim 5.2 we have

$$\begin{aligned}
\mathbf{E}_{\rho}\left(\mathbf{D}\left(\bar{\pi}^l_M \parallel \bar{\mu}^l_M\right)\right) &= \sum_{M \in \mathcal{M}} \rho(M) \mathbf{D}\left(\bar{\pi}^l_M \parallel \bar{\mu}^l_M\right) \\
&= \sum_{M' \in \mathcal{M}^l} \rho_l(M') \sum_{M \in N(M')} \frac{\rho(M)}{\rho_l(M')} \mathbf{D}\left(\bar{\pi}^l_M \parallel \bar{\mu}^l_M\right) \\
&= \sum_{M' \in \mathcal{M}^l} \rho_l(M') \mathbf{V}\left(\bar{\pi}^l_{M'} \parallel \bar{\mu}^l_{M'}\right) \;=\; \mathbf{E}_{\rho_l}\left(\mathbf{V}\left(\bar{\pi}^l_{M'} \parallel \bar{\mu}^l_{M'}\right)\right).
\end{aligned}$$

Since this is true for all l, we can sum up and conclude by Claim 5.1 that

$$\sum_{l=1}^{k} \mathbf{E}_{\rho_l}\left(\mathbf{V}\left(\bar{\pi}^l_{M'} \parallel \bar{\mu}^l_{M'}\right)\right) = \sum_{l=1}^{k} \mathbf{E}_{\rho}\left(\mathbf{D}\left(\bar{\pi}^l_M \parallel \bar{\mu}^l_M\right)\right) \leq -\log \bar{\mu}(A). \qquad \square$$

Since $\mathbf{V}(\vartheta \parallel \psi)$ is always non-negative (see section 3), we can conclude from the last claim that less than $\frac{k}{2}$ coordinates l satisfy

$$\mathbf{E}_{\rho_l}\left(\mathbf{V}\left(\bar{\pi}^l_{M'} \parallel \bar{\mu}^l_{M'}\right)\right) > -\frac{2}{k} \log \bar{\mu}(A).$$

Therefore, more than $\frac{1}{2}k$ coordinates l satisfy

$$\mathbf{E}_{\rho_l}\left(\mathbf{V}\left(\bar{\pi}^l_{M'} \parallel \bar{\mu}^l_{M'}\right)\right) \leq -\frac{2}{k} \log \bar{\mu}(A).$$

By Lemmas 3.3 and 3.5, we also have

$$\sum_{l=1}^{k} \mathbf{D}\left(\bar{\pi}^l \parallel \bar{\mu}^l\right) \leq \mathbf{D}\left(\bar{\pi} \parallel \bar{\mu}\right) = \mathbf{D}\left(\bar{\mu}_A \parallel \bar{\mu}\right) = -\log \bar{\mu}(A)$$

and as before, more than $\frac{1}{2}k$ coordinates l satisfy

$$\mathbf{D}\left(\bar{\pi}^l \parallel \bar{\mu}^l\right) \leq -\frac{2}{k} \log \bar{\mu}(A).$$

Therefore, there exists l with

$$\mathbf{E}_{\rho_l}\left(\mathbf{V}\left(\bar{\pi}^l_{M'} \parallel \bar{\mu}^l_{M'}\right)\right) \le -\frac{2}{k}\log\bar{\mu}(A)$$

and

$$\mathbf{D}\left(\bar{\pi}^l \parallel \bar{\mu}^l\right) \le -\frac{2}{k}\log\bar{\mu}(A).$$

Since $\bar{\mu} = \mu^{\otimes k}$, we have $\bar{\mu}^l_{M'} = \mu$ and $\bar{\mu}^l = \mu$. Thus Lemma 4.2 follows.

6. Families of local protocols. Given X, Y, U, V, Q, every probability measure, $\gamma : X \times Y \to \mathbf{R}^+$, defines a game G_γ. In this section, as before, we denote the value of this game by $w(\gamma)$. The original measure μ is one measure of this type (with value $w(\mu)$). For some finite set D, let $\{\gamma_d\}_{d\in D}$ be a family of measures of this type. Thus, for all $d \in D$, $\gamma_d : X \times Y \to \mathbf{R}^+$ is a probability measure, and has a value $w(\gamma_d)$. Let $p_d \ge 0$ be an arbitrary weight ($\forall d \in D$). We will not always require $\sum_{d\in D} p_d = 1$. However, in the cases that we consider $\sum_{d\in D} p_d$ will be very close to 1. Thus p_d will be "almost" a probability measure on D.

Define

$$w_d = w(\gamma_d), \quad w = \sum_{d\in D} p_d w_d$$

(note that w may be larger than 1, because we didn't require $\sum_{d\in D} p_d = 1$), and the measure $\gamma : X \times Y \to \mathbf{R}^+$ by

$$\gamma = \sum_{d\in D} p_d \gamma_d.$$

In this section we assume that γ is close to μ. We would like to deduce, under some conditions on the measures $\{\gamma_d\}_{d\in D}$, a lower bound for $w(\mu)$ as a function of w, or conversely, an upper bound for w as a function of $w(\mu)$. The conditions on $\{\gamma_d\}_{d\in D}$ will intuitively say that each measure γ_d "locally" describes the behavior of the original measure μ. The goal of the section is to prove Lemma 4.3, which is stated in section 4.

6.1. The basic lemma. First assume that for all $d \in D$, we have sets $X_d \subset X$ and $Y_d \subset Y$, such that

$$\gamma_d = \mu_{X_d \times Y_d}.$$

For all $d \in D$, define

$$a_d = \mu(X_d \times Y_d)$$

and

$$r_d = \frac{\mu\left[(X_d \times (Y - Y_d)) \cup ((X - X_d) \times Y_d)\right]}{\mu(X_d \times Y_d)}.$$

Define

$$r = \sum_{d\in D} p_d r_d.$$

The next lemma gives our basic lower bound for $w(\mu)$ as a function of w.

LEMMA 6.1. *Assume that for some constants $0 \leq \epsilon_0 \leq 1$, $0 \leq \epsilon_1 \leq 1$, we have $\| \gamma - \mu \|_1 \leq \epsilon_0$, and $r \leq \epsilon_1$, and assume (for simplicity) that $w \leq 1$. Let $f : \mathbf{R} \to \mathbf{R}$ be any increasing monotone function such that $f(z) \leq 0$, for $z \leq 0$, and such that for all $0 \leq z \leq 1$, $0 < \delta \leq 1$, we have the inequality*

$$(1-\delta)z + \delta f\left[\frac{1}{\delta}(z-(1-\delta))\right] \geq f(z).$$

Then

$$w(\mu) \geq f(w - 2\sqrt{\epsilon_1} - \epsilon_0).$$

We remark that ϵ_0, ϵ_1 should be thought of as small constants. Some examples for functions f, as above, are given in Corollaries 6.2, 6.3, 6.4. It is always true that for $0 \leq z \leq 1$, $f(z) \leq z$, (to see this, substitute $\delta = 1/2$ to get $f(z) \leq z/2 + f(2z-1)/2$, and by the fact that f is monotone, and $z \leq 1$, we get $f(z) \leq z/2 + f(z)/2$).

Proof. We will first give some intuition. Every measure μ decomposes into its irreducible components. In the simplest case, where $\epsilon_0 = \epsilon_1 = 0$, for all $d : r_d = 0$. Therefore, in this case, $X_d \times Y_d$ is a component of the measure μ (not necessarily irreducible). Since we can further decompose each one of these components, we can assume w.l.o.g. that $X_d \times Y_d$ is irreducible, and therefore that the family $\{\gamma_d\}_{d \in D}$ describes the decomposition of the measure μ into irreducible components. Every protocol for μ defines a protocol for each one of its components. Also, given a protocol for each one of the irreducible components, we can define a protocol for μ (we define the protocol for μ on each irreducible component separately). The value of the protocol for μ is the weighted average of the values of the protocols for the components. Therefore in the special case $\epsilon_0 = \epsilon_1 = 0$, we can simply conclude that $w(\mu) = w$, and the claim follows. In the general case the intuition is basically the same: We will find d_0 such that $X_{d_0} \times Y_{d_0}$ is "almost" a component of μ (i.e., d_0 with small r_{d_0}) and such that w_{d_0} is very close to w. We will concentrate on the set $(X - X_{d_0}) \times (Y - Y_{d_0})$ and continue by induction. Intuitively, we will get a subfamily of $\{\gamma_d\}_{d \in D}$ that will "almost" describe a decomposition of the measure μ into "almost" irreducible components.

First note that

$$\| \gamma \|_1 = \left\| \sum_{d \in D} p_d \gamma_d \right\|_1 = \sum_{d \in D} p_d \| \gamma_d \|_1 = \sum_{d \in D} p_d.$$

Therefore, since $\| \gamma - \mu \|_1 \leq \epsilon_0$ and since $\| \mu \|_1 = 1$, we have by the triangle inequality

$$1 - \epsilon_0 \leq \sum_{d \in D} p_d \leq 1 + \epsilon_0.$$

CLAIM 6.1. *There exists $d_0 \in D$ with*

$$r_{d_0} \leq \sqrt{\epsilon_1}$$

and

$$w_{d_0} \geq w - \sqrt{\epsilon_1} - \epsilon_0.$$

Proof. Denote

$$Z = \{d \in D \mid r_d \geq \sqrt{\epsilon_1}\}.$$

Then,

$$\epsilon_1 \geq r = \sum_{d\in D} p_d r_d \geq \sum_{d\in Z} p_d r_d \geq \sqrt{\epsilon_1} \sum_{d\in Z} p_d.$$

Hence,

$$\sum_{d\in Z} p_d \leq \sqrt{\epsilon_1};$$

thus,

$$w = \sum_{Z} p_d w_d + \sum_{D-Z} p_d w_d \leq \sum_{Z} p_d 1 + \sum_{D-Z} p_d w_d \leq \sqrt{\epsilon_1} + \sum_{D-Z} p_d w_d$$

and therefore,

$$\sum_{D-Z} p_d w_d \geq w - \sqrt{\epsilon_1}.$$

Let $d_0 \in D - Z$ be the element with the maximal w_{d_0}. Then,

$$w_{d_0} \sum_{D-Z} p_d \geq \sum_{D-Z} p_d w_d \geq w - \sqrt{\epsilon_1},$$

and by $\sum_{D-Z} p_d \leq \sum_{d\in D} p_d \leq 1 + \epsilon_0$, we have

$$w_{d_0} \geq (w - \sqrt{\epsilon_1})/(1 + \epsilon_0) \geq (w - \sqrt{\epsilon_1})(1 - \epsilon_0) \geq w - \sqrt{\epsilon_1} - \epsilon_0$$

(since $w \leq 1$).

This completes the proof of Claim 6.1. □

Fix d_0 from the last claim. Define

$$X' = X - X_{d_0}, \quad Y' = Y - Y_{d_0}, \quad \delta = \mu(X' \times Y').$$

Assume that $\delta > 0$. For all $d \in D$ define

$$X'_d = X_d \cap X', \quad Y'_d = Y_d \cap Y', \quad \gamma'_d = \mu_{X'_d \times Y'_d}, \quad w'_d = w(\gamma'_d), \quad a'_d = \mu(X'_d \times Y'_d),$$
$$r'_d = \frac{\mu\left[(X'_d \times (Y' - Y'_d)) \cup ((X' - X'_d) \times Y'_d)\right]}{\mu(X'_d \times Y'_d)}, \quad p'_d = p_d \frac{1}{\delta} \frac{a'_d}{a_d}$$

(recall that $\frac{0}{0}$ is defined to be 0). Notice that r'_d can be ∞, but then $p'_d = 0$. Define

$$r' = \sum_{D} p'_d r'_d, \quad w' = \sum_{D} p'_d w'_d, \quad \gamma' = \sum_{D} p'_d \gamma'_d.$$

Clearly,

$$\gamma'_d(x,y) = \begin{cases} \dfrac{a_d}{a'_d}\gamma_d(x,y) & \text{for } (x,y) \in X' \times Y', \\ 0 & \text{for } (x,y) \notin X' \times Y'. \end{cases}$$

Therefore, by the definitions,

$$\gamma'(x,y) = \begin{cases} \dfrac{1}{\delta}\gamma(x,y) & \text{for } (x,y) \in X' \times Y', \\ 0 & \text{for } (x,y) \notin X' \times Y'. \end{cases}$$

So

$$\gamma' = \left(\frac{1}{\delta}\gamma(X' \times Y')\right)\gamma_{X' \times Y'}.$$

Therefore, we also have

$$\sum_D p'_d \ = \parallel \gamma' \parallel_1 = \ \gamma(X' \times Y')/\delta.$$

We would like to use induction (of the lemma), with the family $\{\gamma'_d\}_{d \in D}$, to deduce a lower bound for $w(\mu_{X' \times Y'})$. In order to use the lemma, we need bounds for r', $\parallel \gamma' - \mu_{X' \times Y'} \parallel_1$, and for w'.
First note that

$$\begin{aligned} r'_d a'_d &= \mu\left[(X'_d \times (Y' - Y'_d)) \cup ((X' - X'_d) \times Y'_d)\right] \\ &\leq \mu\left[(X_d \times (Y - Y_d)) \cup ((X - X_d) \times Y_d)\right] \ = \ r_d a_d. \end{aligned}$$

Therefore, defining

$$\epsilon'_1 = \frac{\epsilon_1}{\delta}$$

we have

$$r' = \sum_D p'_d r'_d = \frac{1}{\delta}\sum_D p_d \frac{a'_d}{a_d} r'_d \leq \frac{1}{\delta}\sum_D p_d r_d = \frac{r}{\delta} \leq \frac{\epsilon_1}{\delta} = \epsilon'_1.$$

Also, define

$$\epsilon'_0 = \parallel \gamma' - \mu_{X' \times Y'} \parallel_1;$$

then clearly,

$$\begin{aligned} \delta\epsilon'_0 = \parallel \delta\gamma' - \delta\mu_{X' \times Y'} \parallel_1 \ &= \ \parallel \gamma(X' \times Y')\gamma_{X' \times Y'} - \mu(X' \times Y')\mu_{X' \times Y'} \parallel_1 \\ = \sum_{X' \times Y'} |\gamma(x,y) - \mu(x,y)| \leq \sum_{X \times Y} |\gamma(x,y) - \mu(x,y)| \ &= \parallel \gamma - \mu \parallel_1 = \epsilon_0. \end{aligned}$$

Thus we have bounds for r',$\parallel \gamma' - \mu_{X' \times Y'} \parallel_1$. The bound for w' will follow from the following two claims.

Claim 6.2.

$$\parallel \gamma \parallel_1 - \delta \parallel \gamma' \parallel_1 \leq \ \epsilon_0 - \delta\epsilon'_0 + (1 - \delta).$$

Proof. Define $Z = (X \times Y) - (X' \times Y')$, and $Z' = X' \times Y'$. Then $\gamma = \gamma(Z)\gamma_Z + \gamma(Z')\gamma_{Z'}$, and $\mu = \mu(Z)\mu_Z + \mu(Z')\mu_{Z'}$. Therefore,

$$\parallel \gamma \parallel_1 = \parallel \gamma(Z)\gamma_Z \parallel_1 + \parallel \gamma(Z')\gamma_{Z'} \parallel_1 = \parallel \gamma(Z)\gamma_Z \parallel_1 + \delta \parallel \gamma' \parallel_1 . \tag{4}$$

In addition,

$$\begin{aligned} \epsilon_0 \geq \parallel \gamma - \mu \parallel_1 &= \parallel \gamma(Z)\gamma_Z - \mu(Z)\mu_Z \parallel_1 + \parallel \gamma(Z')\gamma_{Z'} - \mu(Z')\mu_{Z'} \parallel_1 \\ &= \parallel \gamma(Z)\gamma_Z - \mu(Z)\mu_Z \parallel_1 + \parallel \delta\gamma' - \delta\mu_{Z'} \parallel_1 = \parallel \gamma(Z)\gamma_Z - \mu(Z)\mu_Z \parallel_1 + \delta\epsilon'_0. \end{aligned}$$

Thus, by the triangle inequality,

$$\epsilon_0 - \delta\epsilon_0' \geq \| \gamma(Z)\gamma_Z - \mu(Z)\mu_Z \|_1 \geq \| \gamma(Z)\gamma_Z \|_1 - \| \mu(Z)\mu_Z \|_1 = \| \gamma(Z)\gamma_Z \|_1 - (1-\delta),$$

so,

$$\| \gamma(Z)\gamma_Z \|_1 \leq \epsilon_0 - \delta\epsilon_0' + (1-\delta).$$

The proof follows by substituting this in (4). □

CLAIM 6.3.

$$w' \geq (w - (1-\delta) - \epsilon_0 + \delta\epsilon_0')/\delta.$$

Proof. Any protocol for the game with the measure $\mu_{X_d \times Y_d}$ defines a corresponding protocol for the game with the measure $\mu_{X_d' \times Y_d'}$. Take the best protocol for $\mu_{X_d \times Y_d}$. This protocol for $\mu_{X_d \times Y_d}$ satisfies Q on a set of points of μ-measure $w_d a_d$. At most μ-measure of $a_d - a_d' = \mu(X_d \times Y_d) - \mu(X_d' \times Y_d')$ is outside $X_d' \times Y_d'$. Therefore, at least μ-measure of $w_d a_d - (a_d - a_d')$ is inside. Thus, the corresponding protocol for $\mu_{X_d' \times Y_d'}$ satisfies Q on a set of points of μ-measure $w_d a_d - (a_d - a_d')$ (at least). But this cannot be more than $w_d' a_d'$; thus,

$$w_d' a_d' \geq w_d a_d - a_d + a_d'.$$

Therefore,

$$p_d' w_d' = p_d \frac{1}{\delta} \frac{a_d'}{a_d} w_d' \geq p_d \frac{1}{\delta} \frac{1}{a_d} [w_d a_d - a_d + a_d'] = \frac{1}{\delta} p_d w_d - \frac{1}{\delta} p_d + p_d',$$

and we have

$$\begin{aligned} w' &= \sum_D p_d' w_d' \geq \frac{1}{\delta} \sum_D p_d w_d - \frac{1}{\delta} \sum_D p_d + \sum_D p_d' \\ &= \frac{1}{\delta} w - \frac{1}{\delta} \| \gamma \|_1 + \| \gamma' \|_1 = \frac{1}{\delta} [w - (\| \gamma \|_1 - \delta \| \gamma' \|_1)], \end{aligned}$$

and by Claim 6.2

$$w' \geq (w - (1-\delta) - \epsilon_0 + \delta\epsilon_0')/\delta. \qquad \square$$

By Claim 6.3, we can conclude

$$\begin{aligned} w' - 2\sqrt{\epsilon_1'} - \epsilon_0' &\geq \frac{1}{\delta}(w - (1-\delta) - \epsilon_0 + \delta\epsilon_0') - 2\sqrt{\frac{\epsilon_1}{\delta}} - \epsilon_0' \\ &= \frac{1}{\delta}(w - (1-\delta) - 2\sqrt{\delta}\sqrt{\epsilon_1} - \epsilon_0) \geq \frac{1}{\delta}(w - (1-\delta) - 2\sqrt{\epsilon_1} - \epsilon_0). \end{aligned}$$

Now, we can use induction and apply the lemma for $\mu_{X' \times Y'}$, with the family $\{\gamma_d'\}_{d \in D}$ to get a lower bound

$$w(\mu_{X' \times Y'}) \geq f\left[w' - 2\sqrt{\epsilon_1'} - \epsilon_0'\right] \geq f\left[\frac{1}{\delta}(w - 2\sqrt{\epsilon_1} - \epsilon_0 - (1-\delta))\right]$$

(since f is monotone). Define $z = w - 2\sqrt{\epsilon_1} - \epsilon_0$ to get

$$w(\mu_{X' \times Y'}) \geq f\left[\frac{1}{\delta}(z - (1-\delta))\right].$$

Recall that $w_{d_0} \geq w - \sqrt{\epsilon_1} - \epsilon_0$. Since $a_{d_0} + r_{d_0} a_{d_0} + \delta = \mu(X \times Y) = 1$, we have $a_{d_0} = (1-\delta)/(1+r_{d_0})$. But $r_{d_0} \leq \sqrt{\epsilon_1}$, and therefore

$$a_{d_0} \geq (1-\delta)/(1+\sqrt{\epsilon_1}) \geq (1-\delta)(1-\sqrt{\epsilon_1})$$

and we have

$$a_{d_0} w_{d_0} \geq (1-\delta)(1-\sqrt{\epsilon_1})(w - \sqrt{\epsilon_1} - \epsilon_0) \geq (1-\delta)(w - 2\sqrt{\epsilon_1} - \epsilon_0) = (1-\delta)z.$$

Recall that $X_{d_0} \cap X' = \emptyset$, and $Y_{d_0} \cap Y' = \emptyset$. Given the best protocol for $\mu_{X_{d_0} \times Y_{d_0}}$ and the best protocol for $\mu_{X' \times Y'}$, define a protocol for μ that behaves like the first one on $X_{d_0} \times Y_{d_0}$ and like the other one on $X' \times Y'$. This protocol satisfies Q on a set of points of μ-measure $\mu(X_{d_0} \times Y_{d_0}) w(\mu_{X_{d_0} \times Y_{d_0}}) + \mu(X' \times Y') w(\mu_{X' \times Y'})$ (at least), and therefore proves that

$$\begin{aligned} w(\mu) &\geq a_{d_0} w_{d_0} + \delta w(\mu_{X' \times Y'}) \\ &\geq (1-\delta)z + \delta f\left[\frac{1}{\delta}(z - (1-\delta))\right] \geq f(z) = f(w - 2\sqrt{\epsilon_1} - \epsilon_0), \end{aligned}$$

which proves the lemma. The assumption $z \leq 0 \Rightarrow f(z) \leq 0$ is needed for the base case of the induction. We remark that if $\delta = 0$ the lemma is proved simply from Claim 6.1 or by a continuity argument. □

COROLLARY 6.2. *Under the assumptions of Lemma* 6.1,

$$w(\mu) \geq f(w - 2\sqrt{\epsilon_1} - \epsilon_0),$$

where f is defined by

$$f(z) = \begin{cases} \frac{1}{2}z^2 & \text{for } 0 \leq z \leq 1, \\ 0 & \text{for } z \leq 0. \end{cases}$$

Proof. We just have to prove the inequality (as in Lemma 6.1) for f.

Case a. $z \geq 1 - \delta$.

In this case

$$\begin{aligned} (1-\delta)z + \delta f\left[\frac{1}{\delta}(z-(1-\delta))\right] - f(z) &= (1-\delta)z + \frac{1}{2\delta}(z-(1-\delta))^2 - \frac{1}{2}z^2 \\ &= \frac{1}{2\delta}\left[2\delta(1-\delta)z + z^2 - 2(1-\delta)z + (1-\delta)^2 - \delta z^2\right] \\ &= \frac{1}{2\delta}\left[z^2(1-\delta) - 2z(1-\delta)^2 + (1-\delta)^2\right] \\ &\geq \frac{1-\delta}{2\delta}\left[z^2 - 2z(1-\delta) + (1-\delta)^2\right] \\ &= \frac{1-\delta}{2\delta}\left[z-(1-\delta)\right]^2 \quad \geq \quad 0. \end{aligned}$$

Case b. $z \leq 1 - \delta$.

In this case

$$\begin{aligned} (1-\delta)z + \delta f\left[\frac{1}{\delta}(z-(1-\delta))\right] - f(z) &= (1-\delta)z + 0 - \frac{1}{2}z^2 \\ &\geq z^2 - \frac{1}{2}z^2 = \frac{1}{2}z^2 \geq 0. \end{aligned}$$

□

It is sometimes convenient to denote $v(\mu) = 1 - w(\mu)$, $v = 1 - w$, and $g(t) = 1 - f(1-t)$. In these notations the inequality

$$(1-\delta)z + \delta f\left[\frac{1}{\delta}(z-(1-\delta))\right] - f(z) \geq 0$$

(for $0 \leq z \leq 1$, $0 < \delta \leq 1$), is equivalent (by setting $t = 1 - z$) to

$$(1-\delta)t + \delta g(t/\delta) \leq g(t)$$

(for $0 \leq t \leq 1$, $0 < \delta \leq 1$).

COROLLARY 6.3. *Under the assumptions of Lemma* 6.1, *if* $g : \mathbf{R} \to \mathbf{R}$ *is an increasing monotone function such that* $g(t) \geq 1$, *for* $t \geq 1$, *and such that for all* $0 \leq t \leq 1$, $0 < \delta \leq 1$, *we have*

$$(1-\delta)t + \delta g(t/\delta) - g(t) \leq 0;$$

then,

$$v(\mu) \leq g(v + 2\sqrt{\epsilon_1} + \epsilon_0).$$

COROLLARY 6.4. *Under the assumptions of Lemma* 6.1,

$$v(\mu) \leq 2\sqrt{v + 2\sqrt{\epsilon_1} + \epsilon_0}.$$

Proof. Take $g(t) = 2\sqrt{t}$, then for $0 \leq t \leq 1$, $0 < \delta \leq 1$,

$$\begin{aligned}(1-\delta)t + \delta g(t/\delta) - g(t) &= (1-\delta)t + 2\sqrt{\delta}\sqrt{t} - 2\sqrt{t} \\ &= (1+\sqrt{\delta})(1-\sqrt{\delta})\sqrt{t}\sqrt{t} - 2(1-\sqrt{\delta})\sqrt{t} \\ &= (1-\sqrt{\delta})\sqrt{t}[(1+\sqrt{\delta})\sqrt{t} - 2] \quad \leq \quad 0\end{aligned}$$

(because $(1+\sqrt{\delta})\sqrt{t} - 2 \leq 2 - 2 = 0$). □

It will be convenient to define the function $W_1 : \mathbf{R} \to \mathbf{R}$ in the following way:

$$W_1(z) = \sup_f f(z)$$

where the supremum is taken over all the strictly increasing monotone functions, satisfying the conditions as in Lemma 6.1. Conversely, define $W_2 : \mathbf{R} \to \mathbf{R}$ by

$$W_2(z) = \inf_f f^{-1}(z)$$

where the infimum is taken over the same family of functions.

Take, for convenience, $\epsilon_0 = \epsilon_1 = \epsilon$, and assume for convenience that $w \leq 1$. Lemma 6.1 can now be restated in the following way.

COROLLARY 6.5. *If for a family* $\{\gamma_d\}_{d\in D}$ *we have* $\| \gamma - \mu \|_1 \leq \epsilon$ *and* $r \leq \epsilon$, *then*

$$w(\mu) \geq W_1(w - O(\sqrt{\epsilon}))$$

or, conversely,

$$w \leq W_2(w(\mu)) + O(\sqrt{\epsilon}).$$

By Corollaries 6.2 and 6.4 we have

$$0 < z \Rightarrow 0 < W_1(z), \quad \lim_{z\to 1} W_1(z) = 1, \quad \lim_{z\to 0} W_2(z) = 0, \quad z < 1 \Rightarrow W_2(z) < 1$$

(where the last fact is the important one for us). Also, since f was monotone, $W_1(z), W_2(z)$ are both monotone. Thus, W_1, W_2 are "well behaved." In this manuscript we will not investigate their exact behavior.

6.2. Characterization of measures ϑ, with small $\mathbf{V}(\vartheta \parallel \mu)$. Lemma 4.3 talks about measures γ_d, with small $\mathbf{V_X}(\gamma_d \parallel \mu)$ and small $\mathbf{V_Y}(\gamma_d \parallel \mu)$. We would like to prove this lemma by a reduction to Lemma 6.1. In order to do that, we need one more lemma that gives a characterization of measures ϑ, with small $\mathbf{V_X}(\vartheta \parallel \mu)$ and small $\mathbf{V_Y}(\vartheta \parallel \mu)$. Again, $\mu : X \times Y \to \mathbf{R}^+$ and $\vartheta : X \times Y \to \mathbf{R}^+$ are probability measures.

LEMMA 6.6. *If* $\mathbf{V_X}(\vartheta \parallel \mu), \mathbf{V_Y}(\vartheta \parallel \mu) \leq \epsilon$ *then, for some* m*, there exist partitions* $X = \bigcup_{i=1}^m X_i$, $Y = \bigcup_{i=1}^m Y_i$*, and positive weights* $\{q_i\}_{i=1}^m$ *(i.e., for all* $i : q_i \geq 0$*), such that the function* $h : X \times Y \to \mathbf{R}$*, defined by*

$$h(x,y) = \sum_{i=1}^{m} q_i \mu_{X_i \times Y_i}(x,y)$$

satisfies

$$\parallel \vartheta - h \parallel_1 \leq O(\epsilon^{1/8})$$

and such that

$$\sum_{i=1}^{m} q_i r_i \leq O(\epsilon^{1/8}),$$

where

$$r_i = \frac{\mu\left[(X_i \times (Y - Y_i)) \cup ((X - X_i) \times Y_i)\right]}{\mu(X_i \times Y_i)}.$$

Proof. Assume that $\vartheta(x), \vartheta(y), \mu(x), \mu(y)$ take only strictly positive values (otherwise, just add small constants and normalize). This is done only to simplify the notations. Assume that ϵ is small enough ($\epsilon < 2^{-10}$ is enough), (for $\epsilon \geq 2^{-10}$, $O(\epsilon^{1/8}) = O(1)$, thus $h = \mu$ does the job).

The first claim describes the basic structure of the measure ϑ.

CLAIM 6.4.

$$\left\| \vartheta(x,y) - \frac{\vartheta(x)}{\mu(x)}\mu(x,y) \right\|_1 \leq \sqrt{2\ln 2}\sqrt{\epsilon},$$

$$\left\| \vartheta(x,y) - \frac{\vartheta(y)}{\mu(y)}\mu(x,y) \right\|_1 \leq \sqrt{2\ln 2}\sqrt{\epsilon}.$$

Proof. We will prove the first inequality. The second can be proved in a similar manner.

By Lemma 3.4 we have

$$\begin{aligned}\sum_{x \in X} \vartheta(x) \left(\parallel \vartheta(x,\cdot) - \mu(x,\cdot) \parallel_1\right)^2 &\leq (2\ln 2) \sum_{x \in X} \vartheta(x) \mathbf{D}\left(\vartheta(x,\cdot) \parallel \mu(x,\cdot)\right) \\ &= (2\ln 2)\mathbf{V_X}(\vartheta \parallel \mu) \leq (2\ln 2)\epsilon.\end{aligned}$$

Since for any random variable z: $(\mathbf{E}(z))^2 \leq \mathbf{E}(z^2)$, we have

$$\sum_{x \in X} \vartheta(x) \left(\parallel \vartheta(x,\cdot) - \mu(x,\cdot) \parallel_1\right) \leq \sqrt{2\ln 2}\sqrt{\epsilon},$$

but

$$\sum_{x\in X}\vartheta(x)\left(\parallel \vartheta(x,\cdot)-\mu(x,\cdot)\parallel_1\right) = \sum_{x\in X}\vartheta(x)\sum_{y\in Y}\left|\frac{\vartheta(x,y)}{\vartheta(x)}-\frac{\mu(x,y)}{\mu(x)}\right|$$
$$= \sum_{X\times Y}\left|\vartheta(x,y)-\frac{\vartheta(x)}{\mu(x)}\mu(x,y)\right| = \left\|\,\vartheta(x,y)-\frac{\vartheta(x)}{\mu(x)}\mu(x,y)\,\right\|_1$$

and the claim follows. □

Define

$$R_X(x)=\frac{\vartheta(x)}{\mu(x)},\quad R_Y(y)=\frac{\vartheta(y)}{\mu(y)},\quad R(x,y)=\frac{R_Y(y)}{R_X(x)}$$

(note that $R(x,y)$ is asymmetric). By our assumption, these values are always well defined and strictly positive. Define

$$h_1(x,y)=R_X(x)\mu(x,y),\quad h_2(x,y)=R_Y(y)\mu(x,y).$$

We proved

$$\parallel \vartheta-h_1\parallel_1\le\sqrt{2\ln 2}\sqrt{\epsilon},\quad \parallel \vartheta-h_2\parallel_1\le\sqrt{2\ln 2}\sqrt{\epsilon}.$$

Therefore, by the triangle inequality we also have

$$\parallel h_1\parallel_1\le\parallel\vartheta\parallel_1+\parallel\vartheta-h_1\parallel_1\le\ 1+\sqrt{2\ln 2}\sqrt{\epsilon}.$$

Similarly,

$$\parallel h_2\parallel_1\le\ 1+\sqrt{2\ln 2}\sqrt{\epsilon}$$

and

$$\parallel h_1-h_2\parallel_1\le\parallel\vartheta-h_1\parallel_1+\parallel\vartheta-h_2\parallel_1\le\ 2\sqrt{2\ln 2}\sqrt{\epsilon}.$$

The last inequality shows that most of the measure μ is concentrated on pairs (x,y), with $R_X(x)$ very close to $R_Y(y)$. In the simplest case, where $\epsilon=0$, the entire measure μ is concentrated on pairs, with $R_X(x)=R_Y(y)$. In this case we can partition X according to $R_X(x)$, and Y according to $R_Y(y)$, and define the weight of a subset as $R_X(x)$ (or $R_Y(y)$). The lemma follows then from Claim 6.4 and from the last inequality. In the general case the intuition will be the same, but we will have to partition X and Y into subsets, according to the value of $R_X(x)$ and $R_Y(y)$, where each subset allows small deviations in the value.

Denote

$$\epsilon_1=2\sqrt{2\ln 2}\sqrt{\epsilon}$$

and define

$$Z=\{(x,y)\in X\times Y\ \mid\ |1-R(x,y)|\ge\sqrt{\epsilon_1}\}.$$

Claim 6.5.

$$\sum_Z h_1(x,y)\le\sqrt{\epsilon_1},\quad \sum_Z h_2(x,y)\le\sqrt{\epsilon_1}.$$

Proof. By the definitions

$$\| h_1(x,y)\,(1-R(x,y)) \|_1 = \| h_1 - h_2 \|_1 \leq 2\sqrt{2\ln 2}\sqrt{\epsilon} = \epsilon_1.$$

Therefore,

$$\sum_Z h_1(x,y)\sqrt{\epsilon_1} \leq \sum_Z h_1(x,y)\,|1-R(x,y)| \leq \sum_{X,Y} h_1(x,y)\,|1-R(x,y)| \leq \epsilon_1.$$

Thus,

$$\sum_Z h_1(x,y) \leq \sqrt{\epsilon_1}.$$

The second inequality is proved in the same way. □

Denote

$$\epsilon_2 = \sqrt{-\ln(1-\sqrt{\epsilon_1})}.$$

Then since $\epsilon < 2^{-10}$,

$$\epsilon_2 \leq \sqrt{\ln(1+2\sqrt{\epsilon_1})} \leq \sqrt{2\sqrt{\epsilon_1}} \leq 2\epsilon^{1/8}.$$

For $(x,y) \notin Z$, we have

$$1-\sqrt{\epsilon_1} \leq R(x,y) \leq 1+\sqrt{\epsilon_1}.$$

Hence,

$$\ln(1-\sqrt{\epsilon_1}) \leq \ln R(x,y) \leq \ln(1+\sqrt{\epsilon_1}) \leq -\ln(1-\sqrt{\epsilon_1}).$$

Thus, $(x,y) \notin Z$ implies

$$-\epsilon_2^2 \leq \ln R(x,y) \leq \epsilon_2^2.$$

Let r be a random variable uniformly distributed in the interval $[0,1]$. For every integer i define

$$X_i(r) = \{x \in X \mid (i-1+r)\epsilon_2 \leq \ln R_X(x) < (i+r)\epsilon_2\},$$
$$Y_i(r) = \{y \in Y \mid (i-1+r)\epsilon_2 \leq \ln R_Y(y) < (i+r)\epsilon_2\}.$$

Then, for all r, $\{X_i(r)\}_{i=-\infty}^{\infty}$ is a partition of X, and $\{Y_i(r)\}_{i=-\infty}^{\infty}$ is a partition of Y. Define

$$\hat{Z}(r) = \left\{ (x,y) \in X \times Y \;\middle|\; (x,y) \notin \bigcup_{i=-\infty}^{\infty} X_i \times Y_i \right\}.$$

We will prove that for some r these partitions satisfy the lemma.

Claim 6.6.

$$(x,y) \notin Z \;\Rightarrow\; \Pr_r\left[(x,y) \in \hat{Z}(r)\right] \leq \epsilon_2.$$

Proof. $(x,y) \notin Z$ implies

$$|\ln R_X(x) - \ln R_Y(y)| = |\ln R(x,y)| \le \epsilon_2^2.$$

Assume w.l.o.g. that $\ln R_X(x) \ge \ln R_Y(y)$. For a fixed r, $(x,y) \in X_i(r) \times Y_i(r)$ for some i, unless there exists in the interval $[\ln R_Y(y), \ln R_X(x)]$, a number of the form $(j+r)\epsilon_2$ (for some integer j). The probability for that (over r) is

$$\frac{\ln R_X(x) - \ln R_Y(y)}{\epsilon_2} \le \frac{\epsilon_2^2}{\epsilon_2} = \epsilon_2. \qquad \square$$

CLAIM 6.7.

$$\mathbf{E}_r\left(\sum_{\hat{Z}(r)} h_1(x,y)\right) \le 3\epsilon_2, \quad \mathbf{E}_r\left(\sum_{\hat{Z}(r)} h_2(x,y)\right) \le 3\epsilon_2.$$

Proof. By changing the order of the summations,

$$\begin{aligned}
\mathbf{E}_r\left(\sum_{\hat{Z}(r)} h_1(x,y)\right) &= \sum_{X\times Y} h_1(x,y)\mathrm{Pr}_r\left[(x,y) \in \hat{Z}(r)\right] \\
&= \sum_{X\times Y - Z} h_1(x,y)\mathrm{Pr}_r\left[(x,y) \in \hat{Z}(r)\right] + \sum_{Z} h_1(x,y)\mathrm{Pr}_r\left[(x,y) \in \hat{Z}(r)\right] \\
&\le \sum_{X\times Y - Z} h_1(x,y)\epsilon_2 + \sum_{Z} h_1(x,y)1 \le \epsilon_2 \sum_{X\times Y} h_1(x,y) + \sum_{Z} h_1(x,y) \\
&\le \epsilon_2 \parallel h_1 \parallel_1 + \sqrt{\epsilon_1} \ \le\ \epsilon_2(1 + \sqrt{2\ln 2}\sqrt{\epsilon}) + \sqrt{\epsilon_1} \ \le\ 3\epsilon_2.
\end{aligned}$$

The second inequality is proved in the same way. $\square$

Since $h_1(x,y), h_2(x,y)$ are always positive, we can conclude from the last claim the existence of r_0 with

$$\sum_{\hat{Z}(r)} h_1(x,y) \le 6\epsilon_2, \quad \sum_{\hat{Z}(r)} h_2(x,y) \le 6\epsilon_2.$$

Fix this r_0. Define

$$X_i = X_i(r_0), \quad Y_i = Y_i(r_0), \quad \hat{Z} = \hat{Z}(r_0).$$

We will prove that these partitions satisfy the lemma. Define $h_3 : X \times Y \to \mathbf{R}$ by

$$h_3(x,y) = \begin{cases} h_1(x,y) & \text{for } (x,y) \notin \hat{Z}, \\ 0 & \text{for } (x,y) \in \hat{Z}. \end{cases}$$

Then clearly,

$$\parallel h_1 - h_3 \parallel_1 = \sum_{X,Y} |h_1(x,y) - h_3(x,y)| = \sum_{\hat{Z}} h_1(x,y) \le 6\epsilon_2.$$

By the definitions,

$$h_3(x,y) = \sum_{i=-\infty}^{\infty} R_X(x)\mu(X_i \times Y_i)\mu_{X_i\times Y_i}(x,y).$$

Define

$$q_i = e^{(i-1+r_0)\epsilon_2}\mu(X_i \times Y_i)$$

and

$$h(x,y) = \sum_{i=-\infty}^{\infty} q_i \mu_{X_i \times Y_i}(x,y).$$

CLAIM 6.8.

$$\| h_3 - h \|_1 \leq 2\epsilon_2.$$

Proof. For $x \in X_i$,

$$e^{(i-1+r_0)\epsilon_2} \leq R_X(x) < e^{(i+r_0)\epsilon_2};$$

thus, by the definition of q_i,

$$R_X(x)\mu(X_i \times Y_i) \;\geq\; q_i \;>\; R_X(x)\mu(X_i \times Y_i)e^{-\epsilon_2} \geq R_X(x)\mu(X_i \times Y_i)(1-\epsilon_2).$$

Therefore, for $x \in X_i$

$$0 \;\leq\; R_X(x)\mu(X_i \times Y_i) - q_i \;\leq\; R_X(x)\mu(X_i \times Y_i)\epsilon_2.$$

Since $\mu_{X_i \times Y_i}(x,y) > 0 \Rightarrow x \in X_i$, we have for all x, y,

$$|R_X(x)\mu(X_i \times Y_i) - q_i|\, \mu_{X_i \times Y_i}(x,y) \;\leq\; \epsilon_2 R_X(x)\mu(X_i \times Y_i)\mu_{X_i \times Y_i}(x,y)$$

and therefore,

$$\begin{aligned}
\| h_3 - h \|_1 &= \left\| \sum_{i=-\infty}^{\infty} R_X(x)\mu(X_i \times Y_i)\mu_{X_i \times Y_i}(x,y) - \sum_{i=-\infty}^{\infty} q_i \mu_{X_i \times Y_i}(x,y) \right\|_1 \\
&\leq \left\| \sum_{i=-\infty}^{\infty} |R_X(x)\mu(X_i \times Y_i) - q_i|\, \mu_{X_i \times Y_i}(x,y) \right\|_1 \\
&\leq \left\| \sum_{i=-\infty}^{\infty} \epsilon_2 R_X(x)\mu(X_i \times Y_i)\mu_{X_i \times Y_i}(x,y) \right\|_1 \\
&= \epsilon_2 \| h_3 \|_1 \leq \;\epsilon_2 \| h_1 \|_1 \;\leq\; 2\epsilon_2. \qquad \square
\end{aligned}$$

By the triangle inequality, we can now conclude that

$$\begin{aligned}
\| h - \vartheta \|_1 &\leq \| \vartheta - h_1 \|_1 + \| h_1 - h_3 \|_1 + \| h_3 - h \|_1 \\
&\leq \sqrt{2 \ln 2}\sqrt{\epsilon} + 3\epsilon_2 + 2\epsilon_2 = O(\epsilon^{1/8}).
\end{aligned}$$

Notice that in the definition of h, the sum is actually finite, because X, Y are finite ($\mu(X_i \times Y_i)$ can take a nonzero value only a finite number of times).

CLAIM 6.9.

$$\sum_{i=-\infty}^{\infty} q_i \frac{\mu[X_i \times (Y - Y_i)]}{\mu(X_i \times Y_i)} \leq O(\epsilon^{1/8}),$$

$$\sum_{i=-\infty}^{\infty} q_i \frac{\mu[(X - X_i) \times Y_i]}{\mu(X_i \times Y_i)} \leq O(\epsilon^{1/8}).$$

Proof. Notice that $\mu(X_i \times Y_i)$ can be 0, only if q_i is also 0. As in the previous claim for $x \in X_i$, $q_i \le R_X(x)\mu(X_i \times Y_i)$. Thus,

$$q_i \frac{\mu[X_i \times (Y - Y_i)]}{\mu(X_i \times Y_i)} = \sum_{(x,y)\in X_i\times(Y-Y_i)} \frac{q_i}{\mu(X_i \times Y_i)}\mu(x,y) \le \sum_{(x,y)\in X_i\times(Y-Y_i)} R_X(x)\mu(x,y),$$

and therefore,

$$\sum_{i=-\infty}^{\infty} q_i \frac{\mu[X_i \times (Y - Y_i)]}{\mu(X_i \times Y_i)} \;\le\; \sum_{i=-\infty}^{\infty} \left(\sum_{(x,y)\in X_i\times(Y-Y_i)} R_X(x)\mu(x,y) \right)$$

$$= \sum_{(x,y)\in \hat{Z}} R_X(x)\mu(x,y) \;=\; \sum_{\hat{Z}} h_1(x,y) \;\le\; 6\epsilon_2 \;=\; O(\epsilon^{1/8}).$$

The second inequality is proved in the same way. □

Claim 6.9 and the inequality before give the proof of Lemma 6.6. □

6.3. Proof of Lemma 4.3. Lemma 4.3 will follow as a simple application of Lemmas 6.1 and 6.6. Given ϑ, with

$$\mathbf{V_X}(\vartheta \parallel \mu), \mathbf{V_Y}(\vartheta \parallel \mu) \le \epsilon$$

take m, $\{q_i\}_{i=1}^m$, $\{X_i\}_{i=1}^m$, $\{Y_i\}_{i=1}^m$ from Lemma 6.6. Define

$$\gamma_i = \mu_{X_i \times Y_i}, \quad w_i = w(\gamma_i)$$

and

$$r_i = \frac{\mu[(X_i \times (Y - Y_i)) \cup ((X - X_i) \times Y_i)]}{\mu(X_i \times Y_i)}$$

as in Lemma 6.1. By Lemma 6.6 we have

$$\sum_{i=1}^{m} q_i r_i \;\le\; O(\epsilon^{1/8})$$

and

$$\left\| \vartheta - \sum_{i=1}^{m} q_i \gamma_i \right\|_1 \;\le\; O(\epsilon^{1/8}).$$

Every protocol for ϑ defines also a protocol for each one of the γ_i-s. Therefore, the last inequality also gives

$$w(\vartheta) \;\le\; \sum_{i=1}^{m} q_i w_i + O(\epsilon^{1/8}),$$

which reflects the fact that the function $w(\gamma)$ is concave and has a Lipschitz constant of 1 (see the Introduction).

Lemma 4.3 can be proved now in the following way. Given a family $\{\gamma_d\}_{d\in D}$ of probability measures on $X \times Y$ and weights $\{p_d\}_{d\in D}$, such that $\sum_{d\in D} p_d = 1$, and such that for all $d : p_d \ge 0$, define

$$w_d = w(\gamma_d), \quad w = \sum_{d\in D} p_d w_d, \quad V_d = \mathbf{V}(\gamma_d \parallel \mu).$$

Lemma 4.3 assumes

$$\sum_{d \in D} p_d V_d \ \leq \ \epsilon$$

and

$$\mathbf{D}\left(\sum_{d \in D} p_d \gamma_d \ \middle\| \ \mu \right) \ \leq \ \epsilon,$$

and therefore, by Lemma 3.4

$$\left\| \mu - \sum_{d \in D} p_d \gamma_d \right\|_1 \ \leq \ O(\epsilon^{1/2}).$$

For each measure γ_d, we have by the previous discussion, $\{\gamma_{d,i}\}_{i=1}^{m_d}$, and $\{q_{d,i}\}_{i=1}^{m_d}$, such that

$$\sum_{i=1}^{m_d} q_{d,i} r_{d,i} \ \leq \ O(V_d^{1/8})$$

and

$$\left\| \gamma_d - \sum_{i=1}^{m_d} q_{d,i} \gamma_{d,i} \right\|_1 \ \leq \ O(V_d^{1/8});$$

and as before, we also have

$$w_d \ \leq \ \sum_{i=1}^{m_d} q_{d,i} w_{d,i} + O(V_d^{1/8})$$

(where $r_{d,i}, w_{d,i}$ are defined as before). Define the set $\hat{D}$ by

$$\hat{D} = \{(d,i) \ \mid \ d \in D \ , \ 1 \leq i \leq m_d\}.$$

For each $\hat{d} = (d,i) \in \hat{D}$, define the weight

$$\hat{p}_{(d,i)} = p_d q_{d,i}$$

and the measure

$$\hat{\gamma}_{(d,i)} = \gamma_{d,i}.$$

Look at the family of measures $\{\hat{\gamma}_{(d,i)}\}_{(d,i) \in \hat{D}}$. For this family, define as before

$$\hat{w}_{(d,i)} = w(\hat{\gamma}_{(d,i)}), \quad \hat{w} = \sum_{\hat{d} \in \hat{D}} \hat{p}_{\hat{d}} \hat{w}_{\hat{d}}, \quad \hat{r}_{(d,i)} = r_{d,i}.$$

Then, by the convexity of the function $f(z) = z^{1/8}$, we have

$$\sum_{\hat{d} \in \hat{D}} \hat{p}_{\hat{d}} \hat{r}_{\hat{d}} = \sum_{d \in D} p_d \sum_{i=1}^{m_d} q_{d,i} r_{d,i} \ \leq \ \sum_{d \in D} p_d O(V_d^{1/8}) \ \leq \ O\left(\sum_{d \in D} p_d V_d \right)^{1/8}$$

$$\leq O(\epsilon^{1/8})$$

and

$$\begin{aligned}\left\| \mu - \sum_{\hat{d}\in\hat{D}} \hat{p}_{\hat{d}}\hat{\gamma}_{\hat{d}} \right\|_1 &\le \left\| \mu - \sum_{d\in D} p_d\gamma_d \right\|_1 + \left\| \sum_{d\in D} p_d\gamma_d - \sum_{\hat{d}\in\hat{D}} \hat{p}_{\hat{d}}\hat{\gamma}_{\hat{d}} \right\|_1 \\ &\le O(\epsilon^{1/2}) + \sum_{d\in D} p_d \left\| \gamma_d - \sum_{i=1}^{m_d} q_{d,i}\gamma_{d,i} \right\|_1 \\ &\le O(\epsilon^{1/2}) + \sum_{d\in D} p_d O(V_d^{1/8}) \le O(\epsilon^{1/2}) + O\left(\sum_{d\in D} p_d V_d\right)^{1/8} \\ &\le O(\epsilon^{1/8}),\end{aligned}$$

and also as before,

$$\begin{aligned}\hat{w} = \sum_{\hat{d}\in\hat{D}} \hat{p}_{\hat{d}}\hat{w}_{\hat{d}} &= \sum_{d\in D} p_d \sum_{i=1}^{m_d} q_{d,i}w_{d,i} \ge \sum_{d\in D} p_d \left(w_d - O(V_d^{1/8})\right) \\ &= w - \sum_{d\in D} p_d O(V_d^{1/8}) \\ &\ge w - O(\epsilon^{1/8}).\end{aligned}$$

Now we can apply Corollary 6.5 for the family $\{\hat{\gamma}_{\hat{d}}\}_{\hat{d}\in\hat{D}}$ to get

$$\hat{w} \le W_2(w(\mu)) + O(\epsilon^{1/16})$$

and we can conclude

$$w \le \hat{w} + O(\epsilon^{1/8}) \le W_2(w(\mu)) + O(\epsilon^{1/16}).$$

7. Conclusions. Theorem 1.1 can be generalized using methods introduced herein in many ways. Let us briefly describe two generalizations that seem to follow. The proofs were not verified as carefully as the rest of the paper and should be trusted accordingly. Several other generalizations are described in [42].

7.1. Product of games. Given a game G and a game G', the product game $G \otimes G'$ is defined in the same manner as $G \otimes G$, i.e., if G consists of X, Y, U, V, μ, Q, and G' consists of X', Y', U', V', μ', Q', then the game $G \otimes G'$ consists of the sets $X \times X'$, $Y \times Y'$, $U \times U'$, $V \times V'$ with the measure

$$\mu \otimes \mu'((x,x'),(y,y')) = \mu(x,y)\mu'(x',y')$$

and the predicate

$$Q \otimes Q'((x,x'),(y,y'),(u,u'),(v,v')) = Q(x,y,u,v)Q'(x',y',u',v').$$

In the same way, given k games $G_1, \ldots, G_k$, the product $G_1 \otimes \cdots \otimes G_k$ is defined.

Since in the entire proof of Theorem 1.1 we didn't use the fact that the same game, G, is repeated, and since the function W from Theorem 1.1 is global (and in particular doesn't depend on the game G), the following generalization of Theorem 1.1 follows.

THEOREM 7.1. *Let $W : [0,1] \to [0,1]$ be the function from Theorem* 1.1. *Given k games $G_1, \ldots, G_k$, define*

$$w = MAX[w(G_1), \ldots, w(G_k)],$$

and

$$s = MAX[s(G_1), \ldots, s(G_k), 2].$$

Then

$$w(G_1 \otimes \cdots \otimes G_k) \leq W(w)^{k/\log_2(s)}.$$

As before, $w = \text{MAX}[s(G_1), \ldots, s(G_k), 2]$ can be replaced with

$$w = \text{MAX}[CC(G_1), \ldots, CC(G_k), 2]$$

or with

$$w = \text{MAX}[\rho(G_1), \ldots, \rho(G_k), 2].$$

7.2. Probabilistic predicates. Our second generalization deals with the probabilistic case, where the predicate Q, of a game G, depends also on the random string r (i.e., Q is probabilistic). Without loss of generality we can assume that there exists a second random string, $\tilde{r}$, such that $\tilde{r}$ is independent of r (and, therefore, also of (x, y, u, v)), and such that the predicate Q depends on $x, y, u, v, \tilde{r}$ (and not on the random string r).

As before, the value of a protocol for the game G is defined to be the probability that $Q(x, y, u(x), v(y), \tilde{r}) = 1$, where (x, y) is chosen according to μ. As before, the value of the game, $w(G)$, is defined to be the maximal value of all protocols for G.

THEOREM 7.2. *Let $W : [0,1] \to [0,1]$ be the function from Theorem* 1.1. *Given a probabilistic game G (as above), with value $w(G)$, and answer-size $s(G) \geq 2$:*

$$w(G^{\otimes k}) \leq W(w(G))^{k/\log_2(s(G))}.$$

As before, $\log_2(s(G))$ can be replaced with $CC(G)$, or with $\rho(G)$, defined in the following way: Define $G_{\tilde{r}}$ to be the deterministic game obtained by fixing the second random string to $\tilde{r}$. Then define $CC(G)$ to be the maximum, taken over $\tilde{r}$, of $CC(G_{\tilde{r}})$, and define $\rho(G)$ to be the maximum, taken over $\tilde{r}$, of $\rho(G_{\tilde{r}})$.

Sketch of proof. First we claim that the proof of Theorem 1.2 holds (with minor changes) for probabilistic games as well. In section 4, the fact that the function w_l is concave is used to prove inequality (3) (and the inequality before). The concavity of the value function of a game is proved in the introduction for deterministic games. The same argument, however, holds for probabilistic games. Using this fact, one can verify that the entire argument of section 4 holds for probabilistic games as well. Now, section 4 uses Lemmas 4.2 and 4.3. Lemma 4.2 does not depend on the game G at all. The proof of Lemma 4.3 (in section 6) is based on Lemmas 6.1 and 6.6. Lemma 6.6 does not depend on the game G as well. Therefore, we just have to verify that the proof of Lemma 6.1 holds for probabilistic games. In the proof of Lemma 6.1 we use the fact that the game is deterministic only in the proof of Claim 6.3. It is not hard to see, however, that a probabilistic version of that proof can be given.

Theorem 7.2 is now proved using Theorem 1.2 in the same manner as before (see section 2). The difference is that now, given $z = (x', y', u', v') \in Z$, $\bar{Q}^1$ is still

not determined on the set $A(z)$, because $\bar{Q}^1$ depends also on the second random string corresponding to the first coordinate (denote this random string by $\tilde{r}^1$). In the deterministic case, we disregarded sets $A(z)$ for every z s.t. Q is not satisfied on z. In order to be able to do the same in the probabilistic case, we will have to have a copy of $A(z)$ for every possible $\tilde{r}^1$ (for $\tilde{r}^1 = r'$ denote this copy by $A_{r'}(z)$). We then disregard every copy $A_{r'}(z)$ for every z, r' s.t. Q is not satisfied on z, r'.

A different way to see how Theorem 7.2 is proved using Theorem 1.2 is to define $q(z)$ to be the probability that $Q(z, \tilde{r}) = 1$. (For a deterministic game, $q(z)$ is always 0 or 1). It is not too hard to see that the entire argument of section 2 can be generalized to the case where $0 \leq q(z) \leq 1$.

Alternatively, we can present the proof of Theorem 7.2 in the following way.

We can view a probabilistic game G in the following equivalent way. Player I receives (as an input) the pair $(x, \tilde{r})$ and Player II receives the pair $(y, \tilde{r})$. The protocols of the players are restricted as to depend only on the first input, i.e., x for the first player, and, respectively, y for the second player, (and not on the second input $\tilde{r}$). We call such a protocol a restricted protocol.

The game G is now deterministic because we can think of the predicate as being depended only on the inputs and the answers of the two players. The class of allowed protocols, however, is now a subclass of all possible protocols. We claim that the entire proof of Theorem 1.1 (including the proof of Theorem 1.2) is correct even if we allow only restricted protocols.

In some parts of the proof (e.g., section 4), we start from a protocol, P, for a game, and obtain from this protocol many protocols for many other games. These protocols are obtained either by projection (on one coordinate) or by restriction to a product subset. In other parts (e.g., section 6) a protocol P for a game is composed from other protocols for different games. In order to see that the proof of Theorem 1.1 holds even if we allow only restricted protocols, one should verify that if the original protocols are restricted then every protocol obtained by one of these three methods is also restricted. If the new protocol is obtained by projection of a restricted protocol or by composition of restricted protocols (i.e., by the first or third method), then it is very easy to see that the new protocol is also restricted. If the new protocol is obtained by a restriction of a restricted protocol to a subset, then the new protocol is not necessarily restricted. If that subset does not depend on $\tilde{r}$, however, then the new protocol is restricted. It is not hard to verify that the subsets used (in the proof) never depend on $\tilde{r}$.

Theorem 1.1 is thus correct even if we allow only restricted protocols. Theorem 7.2 follows. □

Acknowledgments. I would like to thank Uri Feige and Moni Naor for presenting the problem to me, and Uri Feige, Peter Hajnal, Joe Kilian, Dieter Van Melkebeek, Mario Szegedy, Endre Szemeredi, Gabor Tardos, Oleg Verbitsky, and Avi Wigderson for helpful and encouraging discussions on the subject and on the content of this paper. I would like to thank the two referees for many helpful comments. This research was carried out while the author was a postdoc at Princeton University and DIMACS.

REFERENCES

[1] N. Alon, *Probabilistic methods in extremal finite set theory*, in Proc. Conference on Extremal Problems for Finite Sets, Bolyai Soc. Math. Stud. 3, János Bolyai Society, Budapest, Hungary, 1994, pp. 39–57.

[2] S. Arora, *Proof Verification and Hardness of Approximation Problems*, Ph.D. dissertation, University of California, Berkeley, CA, http://www.cs.princeton.edu/∼arora, 1994.

[3] S. Arora and C. Lund, *Hardness of approximations*, in Approximation Algorithms for NP-Hard Problems, D. Hochbaum, ed., PWS Publishing, Boston, MA, 1996; also available online from http://www.cs.princeton.edu/∼arora.

[4] S. Arora and S. Safra, *Probabilistic checking of proofs: A new characterization of NP*, in Proc. FOCS 1992, IEEE Computer Society Press, Los Alamitos, CA, pp. 2–13.

[5] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and intractability of approximation problems*, in Proc. FOCS 1992, IEEE Computer Society Press, Los Alamitos, CA, pp. 14–23.

[6] M. Bellare, *Interactive proofs and approximation*, in Proc. Israel Symposium on Theory of Computing and Systems, IEEE Computer Society Press, Los Alamitos, CA, 1993, pp. 266–274.

[7] L. Babai, L. Fortnow, and C. Lund, *Non-deterministic exponential time has two-prover interactive protocols*, in Proc. FOCS 1990, IEEE Computer Society Press, Los Alamitos, CA, pp. 16–25.

[8] M. Bellare, S. Goldwasser, C. Lund, and A. Russell, *Efficient probabilistic checkable proofs and applications to approximation*, in Proc. STOC 1993, ACM, New York, pp. 294–304.

[9] M. Bellare, O. Goldreich, and M. Sudan, *Free bits, PCPs, and nonapproximability—towards tight results*, SIAM J. Comput., 27 (1998), pp. 804–915.

[10] M. Bellare and P. Rogaway, *The complexity of approximating a nonlinear program*, Math. Programming, 69 (1995), pp. 429–441.

[11] M. Bellare and M. Sudan, *Improved non-approximability results*, in Proc. STOC 1994, ACM, New York, pp. 184–193.

[12] M. Ben-or, S. Goldwasser, J. Kilian, and A. Wigderson, *Multi prover interactive proofs: How to remove intractability*, in Proc. STOC 1988, ACM, New York, pp. 113–131.

[13] M. Ben-or, S. Goldwasser, J. Kilian, and A. Wigderson, *Efficient identification schemes using two prover interactive proofs*, in Proc. Crypto 1989, Springer-Verlag, New York, 1990, pp. 498–506.

[14] J. Cai, A. Condon, and R. Lipton, *On bounded round multi-prover interactive proof systems*, in Proc. Structure in Complexity Theory, 1990, IEEE Computer Society Press, Los Alamitos, CA, pp. 45–54.

[15] J. Cai, A. Condon, and R. Lipton, *Playing games of incomplete information*, Theoret. Comput. Sci., 103 (1992), pp. 25–38.

[16] J. Cai, A. Condon, and R. Lipton, *PSPACE is provable by two provers in one round*, J. Comput. System Sci., 48 (1994), pp. 183–193.

[17] I. Csiszar and J. Korner, *Information Theory: Coding Theorems for Discrete Memoryless Systems*, Academic Press, New York, London, 1981.

[18] C. Dwork, U. Feige, J. Kilian, M. Naor, and S. Safra, *Low communication, 2-prover zero-knowledge proofs for NP*, in Proc. Crypto 1992, Springer-Verlag, Berlin, pp. 217–229.

[19] U. Feige, *On the success probability of the two provers in one round proof systems*, in Proc. Structures 1991, pp. 116–123.

[20] U. Feige, *Error Reduction by Parallel Repetition: The State of the Art*, Technical report CS95-32, Weizmann Institute of Science, Rehovot, Israel.

[21] L. Fortnow, *Complexity-Theoretic Aspects of Interactive Proof Systems*, Ph.D. thesis, Report MIT/LCS/TR-447, MIT, Cambridge, MA, 1989.

[22] U. Feige, S. Goldwasser, L. Lovasz, M. Safra, and M. Szegedy, *Approximating clique is almost NP-complete*, in Proc. FOCS 1991, IEEE Computer Society Press, Los Alamitos, CA, pp. 2–12.

[23] U. Feige and J. Kilian, *Two prover protocols: Low error at affordable rates*, in Proc. STOC 1994, ACM, New York, pp. 172–183.

[24] U. Feige and J. Kilian, *Impossibility results for recycling random bits in two prover proof systems*, in Proc. STOC 1995, ACM, New York, pp. 457–468.

[25] U. Feige and L. Lovasz, *Two-prover one-round proof systems, their power and their problems*, in Proc. STOC 1992, ACM, New York, pp. 733–744.

[26] L. Fortnow, J. Rompel, and M. Sipser, *On the power of multi-prover interactive protocols*, in Proc. Structures 1988, pp. 156–161.

[27] U. Feige and O. Verbitsky, *Error reduction by parallel repetition: A negative result*, in Proc. 11th Annual IEEE Conference on Computational Complexity, 1996, IEEE Computer Society Press, Los Alamitos, CA, pp. 70–76.

[28] L. Fortnow, J. Rompel, and M. Sipser, *Errata for "On the power of multi-prover interactive protocols,"* in Proc. Structure in Complexity Theory 1990, IEEE Computer Society Press, Los Alamitos, CA, pp. 318–319.

[29] R. M. Gray, *Entropy and Information Theory*, Springer-Verlag, New York, 1990.
[30] J. Håstad, *Testing of the long code and hardness for clique*, in Proc. STOC 1996, ACM, New York, pp. 11–19.
[31] J. Håstad, *Clique is Hard to Approximate Within* $n^{1-\epsilon}$, in Proc. FOCS 1996, IEEE Computer Society Press, Los Alamitos, CA, pp. 627–636.
[32] J. Håstad, *Some optimal inapproximability results*, in Proc. STOC 1997, ACM, New York, pp. 1–10.
[33] J. Kilian, *Strong Separation Models of Multi Prover Interactive Proofs*, DIMACS Workshop on Cryptography, October 1990.
[34] E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge University Press, London, Cambridge, 1997.
[35] B. Kalyanasundaram and G. Schnitger, *The probabilistic communication complexity of set intersection*, in Proc. Structure in Complexity Theory 1987, IEEE Computer Society Press, Los Alamitos, CA, pp. 41–49.
[36] D. Lapidot and A. Shamir, *A one-round, two-rover, zero-knowledge protocol for NP*, in Combinatorica, 15 (1995), pp. 203–214.
[37] D. Lapidot and A. Shamir, *Fully parallelized multi prover protocols for NEXP-time*, in Proc. FOCS 1991, IEEE Computer Society Press, Los Alamitos, CA, pp. 13–18.
[38] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, in Proc. STOC 1993, ACM, New York, pp. 286–293.
[39] D. Peleg, *On the maximum density of* 0-1 *Matrices with no forbidden rectangles*, Discrete Math., 140 (1995), pp. 269–274.
[40] R. Raz, *A parallel repetition theorem*, in Proc. STOC 1995, ACM, New York, pp. 447–556.
[41] A. A. Razborov, *On the distributional complexity of disjointness*, Theoret. Comput. Sci., 106 (1992), pp. 385–390.
[42] I. Parnafes, R. Raz, and A. Wigderson, *Direct product results and the GCD problem in old and new communication models*, in Proc. STOC 1997, ACM, New York, pp. 363–372.
[43] G. Tardos, *Multi-prover encoding schemes, and three-prover proof systems*, J. Comput. System Sci., 53 (1996), pp. 251–260.
[44] O. Verbitsky, *Towards the parallel repetition conjecture*, Theoret. Comput. Sci., 157 (1996), pp. 277–282.
[45] O. Verbitsky, *The Parallel Repetition Conjecture for Trees is True*, manuscript, 1994.

SIAM J. COMPUT.
Vol. 27, No. 3, pp. 804–915, June 1998
© 1998 Society for Industrial and Applied Mathematics
012

FREE BITS, PCPS, AND NONAPPROXIMABILITY—TOWARDS TIGHT RESULTS*

MIHIR BELLARE†, ODED GOLDREICH‡, AND MADHU SUDAN§

In honor of Shimon Even's 60th birthday

Abstract. This paper continues the investigation of the connection between probabilistically checkable proofs (PCPs) and the approximability of NP-optimization problems. The emphasis is on proving *tight* nonapproximability results via consideration of measures such as the "free-bit complexity" and the "amortized free-bit complexity" of proof systems.

The first part of the paper presents a collection of new proof systems based on a new error-correcting code called the long code. We provide a proof system that has amortized free-bit complexity of $2 + \epsilon$, implying that approximating MaxClique within $N^{\frac{1}{3}-\epsilon}$, and approximating the Chromatic Number within $N^{\frac{1}{5}-\epsilon}$, are hard, assuming NP $\neq$ coRP, for any $\epsilon > 0$. We also derive the first explicit and reasonable constant hardness factors for Min Vertex Cover, Max2SAT, and Max Cut, and we improve the hardness factor for Max3SAT. We note that our nonapproximability factors for MaxSNP problems are appreciably close to the values known to be achievable by polynomial-time algorithms. Finally, we note a general approach to the derivation of strong nonapproximability results under which the problem reduces to the construction of certain "gadgets."

The increasing strength of nonapproximability results obtained via the PCP connection motivates us to ask how far this can go and whether PCPs are inherent in any way. The second part of the paper addresses this. The main result is a "reversal" of the connection due to Feige et al. (FGLSS connection) [*J. ACM*, 43 (1996), pp. 268–292]: where the latter had shown how to translate proof systems for NP into NP-hardness of approximation results for MaxClique, we show how any NP-hardness of approximation result for MaxClique yields a proof system for NP. Roughly, our result says that for any constant f, if MaxClique is NP-hard to approximate within $N^{1/(1+f)}$, then NP $\subseteq \overline{\text{FPCP}}[\log, f]$, the latter being the class of languages possessing proofs of logarithmic randomness and amortized free-bit complexity f. This suggests that PCPs are inherent to obtaining nonapproximability results. Furthermore, the tight relation suggests that reducing the amortized free-bit complexity is necessary for improving the nonapproximability results for MaxClique.

The third part of our paper initiates a systematic investigation of the properties of PCP and FPCP (free PCP) as a function of the following various parameters: randomness, query complexity, free-bit complexity, amortized free-bit complexity, proof size, etc. We are particularly interested in "triviality" results, which indicate which classes are *not* powerful enough to capture NP. We also distill the role of randomized reductions in this area and provide a variety of useful transformations between proof checking complexity classes.

Key words. intractability, approximation, NP-hardness, probabilistic proof systems

AMS subject classification. 68Q15

PII. S0097539796302531

---

*Received by the editors April 24, 1996; accepted for publication (in revised form) August 5, 1997. An extended abstract of this paper appeared in the *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 422–431. It was backed up by the first versions of this large paper [20], which were posted on the *Electronic Colloquium on Computational Complexity (ECCC)* (www.ecc.uni-trier.de/eccc/). The paper went through several revisions due to improvements and corrections in the results. These were regularly posted on ECCC as revisions to [20]. The present paper is the fifth version of the work.

http://www.siam.org/journals/sicomp/27-3/30253.html

†Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093 (mihir@cs.ucsd.edu). The research of this author was supported in part by NSF CAREER Award CCR-9624439 and by a 1996 Packard Foundation Fellowship in Science and Engineering. Some of this work was done while the author was at IBM.

‡Department of Computer Science and Applied Mathematics, Weizmann Institute of Sciences, Rehovot, 76100 Israel (oded@wisdom.weizmann.ac.il). The research of this author was supported in part by grant 92-00226 from the US–Israel Binational Science Foundation (BSF), Jerusalem, Israel.

§Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 (madhu@theory.lcs.mit.edu). Some of this work was done while the author was at IBM.

1. Introduction. In the MaxClique problem, we are given a graph G and must find the value of MaxClique$(G) = \max\{\,|S| : S \text{ is a clique in } G\,\}$. This is an example of an NP-optimization problem, of which others are to find the chromatic number of a graph; the size of the smallest vertex cover, etc. These problems arise in many settings and efficient solutions are much desired. Unfortunately, many important NP-optimization problems (those mentioned above in particular) are NP-hard to solve. So algorithm designers seek efficient (polynomial time) approximation algorithms.

An approximation algorithm delivers a number that is supposed to be close to optimal. The quality of the algorithm is measured in terms of how close this number is to optimal. For example, if $\mu(N) \geq 1$ is a function of the number N of vertices in a graph G, then we say an algorithm A approximates MaxClique within μ, or is a factor μ approximation algorithm, if MaxClique$(G)/\mu(N) \leq A(G) \leq$ MaxClique(G) for every graph G. (For a minimization problem such as Chromatic Number, we require instead that ChromNum$(G) \leq A(G) \leq \mu(N) \cdot$ ChromNum(G), where ChromNum(G) is the chromatic number of G.)

The search for efficient approximation algorithms achieving good factors has met with varied success. For some problems, good approximation algorithms were found. For some important problems, including MaxClique and Chromatic Number, the best approximation algorithms found achieved factors only marginally better than the trivial factor of N. For others, like Minimum Vertex Cover, simple algorithms achieving reasonable factors were discovered quite quickly, but it was unclear whether one could do better. Algorithm designers want to know whether this is due to some inherent intractability or only to the lack of cleverness in algorithm design.

Some early nonapproximability results were able to indicate (at least for some problems) that very good approximation (i.e., achieving factors very close to optimal) can be NP-hard. But the real breakthrough came more recently when a strong hardness of approximation result for MaxClique was shown by establishing a connection between MaxClique and the existence of probabilistically checkable proof (PCP) systems for NP. Since then, similar connections have been found to other optimization problems. Meanwhile, with the construction of more efficient proof systems, the factors within which approximation is shown hard continue to increase. Indeed, in some cases, even tight results seem in sight.

This paper continues the development of the connection between PCPs and hardness of approximation with the goal of getting tight results. On the one hand, we continue past work by building new proof systems and obtaining improved nonapproximability results; on the other hand we open some new directions with an exploration of the limits of the PCP connection.

In what follows we provide a little background and then a high level overview of our results. The rich history of the ideas in this area is overviewed in section 1.3, and more detailed histories are provided in the body of the paper.

1.1. Some background and definitions. We will be informal and as brief as possible; formal definitions can be found in section 2.

Proof systems and parameters. A probabilistic proof system is described by a probabilistic, polynomial time *verifier* V. It takes an input x of length n and tosses coins R. It has oracle access to a poly(n) length string σ describing the proof; to access a bit it writes an $O(\log n)$ bit address and the corresponding bit of the proof is returned. Following its computation, it will either accept or reject its input x. The accepting probability, denoted $\mathtt{ACC}[\,V(x)\,]$, is the maximum, over all σ, of the probability (over R) that V accepts x on coins R and proof string σ. While the task is typically language recognition (namely, to recognize whether x is in some fixed

language L) we will, more generally, consider promise problems (A, B) consisting of a set A of "positive" instances and a set B of "negative" instances [36]. A language L is identified with the promise problem $(L, \overline{L})$.

Of interest in our applications are various parameters of the system. The completeness probability $c = c(n)$ and the soundness probability $s = s(n)$ are defined in the usual ways. In case $c = 1$, we say that the system has *perfect completeness*. The gap is $g = c/s$. The query complexity is the maximum (over all coin tosses and proof strings) of the number of bits of the proof that are examined by the verifier. The free-bit complexity, roughly speaking, is the logarithm of number of possible accepting configurations of V on coins R and input x. (For example, a verifier which makes three queries and accepts, iff the parity of the answers is odd, has four accepting configuration and thus free-bit complexity 2.)

Either the query or the free-bit complexity may be considered in amortized form, for example, the amortized free-bit complexity is the free-bit complexity (of a proof system with perfect completeness) divided by the logarithm of the gap (i.e., the number of free bits needed per factor of 2 increases in the gap). Also, either the query or free-bit complexity may be considered on the average, the average being over the random string of the verifier.

Denote by $\mathrm{PCP}_{c,s}[r, q]$ the class of promise problems recognized by verifiers tossing r coins, having query complexity q, and achieving completeness probability c and soundness probability s. The class $\mathrm{FPCP}_{c,s}[r, f]$ is defined analogously with f being the free-bit complexity, $\overline{\mathrm{PCP}}[r, q]$ is defined analogously with q being the amortized query complexity, and $\overline{\mathrm{FPCP}}[r, f]$ is defined analogously with f being the amortized free-bit complexity.

MaxClique approximation. Although we look at many optimization problems there is a particular focus on MaxClique. Recall that the best known polynomial-time approximation algorithm for MaxClique achieves a factor of only $N^{1-o(1)}$ [28], scarcely better than the trivial factor of N. (Throughout the paper, when discussing the MaxClique problem or any other problem about graphs, N denotes the number of vertices in the graph.) Does there exist an $N^{1-\epsilon}$ factor approximation algorithm for MaxClique for some $\epsilon > 0$? An additional motivation for searching for such "weak" approximation algorithms was suggested by Blum [26]. He showed that a polynomial-time $N^{1-\epsilon}$-factor approximation algorithm for MaxClique implies a polynomial-time algorithm to color a three colorable graph with $O(\log N)$ colors [26], which is much better than what is currently known [63]. But perhaps $N^{1-o(1)}$ is the best possible. Resolving the approximation complexity of this basic problem seems, in any case, to be worth some effort.

Gaps in clique size. Hardness of approximation (say of MaxClique) is typically shown via a reduction to a promise problem with gaps in MaxClique size. Specifically, let $\mathsf{Gap}\text{-MaxClique}_{c,s}$ be the promise problem (A, B) defined as follows: A is the set of all graphs G with $\mathrm{MaxClique}(G)/N \geq c(N)$, and B is the set of all graphs G with $\mathrm{MaxClique}(G)/N < s(N)$. The gap is defined as c/s. Now, a hardness result will typically specify a value of the gap $g(N) = c(N)/s(N)$ for which $\mathsf{Gap}\text{-MaxClique}_{c,s}$ is NP-hard under a (randomized) Karp reduction. This means that there is no polynomial-time algorithm to approximate the MaxClique size of an N node graph within $g(N)$, unless NP has randomized polynomial-time algorithms.[1]

[1] Throughout this paper, the latter assumption is formulated by writing NP $\neq$ coRP. We comment that this form is equivalent to NP $\neq$ ZPP, where ZPP denotes the intersection of the classes RP and coRP. Note that if NP is contained in coRP, then coNP $\subseteq$ RP and NP $\subseteq$ coRP $\subseteq$ coNP $\subseteq$ RP follows.

Focus	Error	Queries	Free Bits	Previous Related Result
3 queries	0.85	3	2	error $\frac{72}{73}$ via MaxSAT [23]
2 free bits	0.794	$O(1)$	2	
error 1/2	$\frac{1}{2}$	11	7	32 queries (24 on average) [41]
amortized free bits	$O(2^{-m})$	2^{3m}	$2m$	$3m$ free bits [23]

FIG. 1. *New PCP systems for NP, all with logarithmic randomness.*

Gap problems can be similarly defined for all the other optimization problems we consider. From now on, we discuss approximation in terms of these gap problems.

THE CONNECTION: MAKING GAPS FROM PROOFS. The FGLSS reduction [40] is a reduction of a promise problem (A, B) to Gap-MaxClique$_{c,s}$ for some appropriate c, s defined by the reduction. It works by using a verifier V of a pcp system for (A, B) to map any instance $x \in A \cup B$ to a graph G_x so that MaxClique(G_x) reflects $\mathtt{ACC}[\,V(x)\,]$. For the best results, one typically uses a randomized form of this reduction due to [25, 86] and it is this form that we will assume henceforth.

An NP-hard gap problem is obtained roughly as follows. First, one exhibits an appropriate proof system for NP. Then one applies the FGLSS reduction. The factor indicated hard depends on the proof system parameters. A key element in getting better results has been the distilling of appropriate pcp-parameters. The sequence of works [40, 9, 8, 21, 41, 23] leads us through the following sequence of parameters: query complexity, free-bit complexity and, finally, for the best known results, amortized free-bit complexity. The connection, in terms of amortized free bits, can be stated as follows: if NP reduces to $\overline{\text{FPCP}}[\log, f]$, then NP also reduces to Gap-MaxClique$_{c,s}$, with gap $c(N)/s(N) = N^{1/(1+f)}$. (In both cases, the reduction is via randomized Karp reductions, and terms of $\epsilon > 0$ which can be arbitrarily small are ignored.) In particular if NP $\subseteq \overline{\text{FPCP}}[\log, f]$, then approximating the MaxClique size of an N vertex graph within $N^{1/(1+f)}$ in polynomial time is not possible unless NP has efficient randomized polynomial-time algorithms.

1.2. Overview of our results.

1.2.1. New proof systems and nonapproximability results. This section describes the new proof systems that we construct and the nonapproximability results that we derive from them.

NEW PROOF SYSTEMS. Summarized below and also presented in Figure 1 are several new ways of capturing NP via probabilistic proof systems:

(1) For every $\epsilon > 0$ it is the case that NP $\subseteq \overline{\text{FPCP}}[\,\log,\, 2 + \epsilon\,]$.
(2) NP $\subseteq \text{PCP}_{1,1/2}[\log, 11]$.
(3) NP $\subseteq \text{FPCP}_{1,s}[\log, 2]$ for $s = 0.794$.
(4) NP $\subseteq \text{PCP}_{1,s}[\log, 3]$ for any $s > 0.85$.

As explained below, some of these results are motivated by applications; others are purely interesting items in complexity theory.

The search for proof systems of low amortized free-bit complexity is motivated, of course, by the FGLSS reduction. Bellare and Sudan [23] have shown that NP $\subseteq \overline{\text{FPCP}}[\,\log,\, 3+\epsilon\,]$ for every $\epsilon > 0$. The first result above improves upon this, presenting a new proof system with amortized free-bit complexity $2 + \epsilon$.

The question of how low one can get the (worst-case and average) query complexity required to attain soundness error 1/2 was investigated in earlier works due to their applicability to obtaining MaxClique hardness results. We now know that one can do better with amortized free-bit complexity. Nevertheless, the original question

Problem	Approx		Non-Approx		
	Factor	Due to	Our Factor	Previous Factor	Assumption
Max3SAT	1.258	[85, 51, 84]	1.038	$1+\frac{1}{72}$ [23]	P $\neq$ NP
MaxE3SAT	$1+\frac{1}{7}$	folklore	$1+\frac{1}{26}$	unspecified [8]	P $\neq$ NP
Max2SAT	1.075	[51, 39]	1.013	$1+\frac{1}{504}$ (implied by [23])	P $\neq$ NP
Max$\oplus$SAT	2	folklore	$1+\frac{1}{7}$		P $\neq$ NP
MaxCUT	1.139	[51]	1.014	unspecified [8]	P $\neq$ NP
MinVC	$2-o(1)$	[14, 74]	$1+\frac{1}{15}$	unspecified [8]	P $\neq$ NP
MaxClique	$N^{1-o(1)}$	[28]		$N^{\frac{1}{4}}$ [23]	NP $\not\subseteq$ coR$\tilde{\text{P}}$
			$N^{\frac{1}{3}}$	$N^{\frac{1}{5}}$	coRP $\neq$ NP
			$N^{\frac{1}{4}}$	$N^{\frac{1}{6}}$ [23]	P $\neq$ NP
Chromatic Number	$N^{1-o(1)}$	[28]		$N^{\frac{1}{10}}$ [23]	NP $\not\subseteq$ coR$\tilde{\text{P}}$
			$N^{\frac{1}{5}}$	$N^{\frac{1}{7}}$ [45]	coRP $\neq$ NP
			$N^{\frac{1}{7}}$	$N^{\frac{1}{14}}$ [23]	P $\neq$ NP

FIG. 2. *Approximation factors attainable by polynomial-time algorithms (denoted Approx) versus factors we show are hard to achieve (denoted Non-Approx).* Max$E$3SAT *(resp.,* Max$\oplus$SAT*) denotes the maximization problem for conjunctive normal form (CNF) formulae having exactly three different literals in each clause (resp., a conjunction of parity clauses).*

is still one to which we are curious to know the answer. Our second result above refers to this question.

Minimizing the soundness error obtainable using only two (nonamortized!) free bits is important for a more pragmatic reason. It enables us to get the first explicit and reasonably strong constant nonapproximability result for the Min Vertex Cover problem. This application is discussed below.

Finally, the soundness achievable using only three query bits is natural to consider given the results on the Max3SAT gap problem. Indeed, if there is an NP-hard Max3SAT gap problem with certain gap, then one can easily get a three query proof system with the same gap. However, one can do better as indicated above.

NEW NONAPPROXIMABILITY RESULTS. Our results are summarized in Figure 2. (In the last items, we ignore terms of N^{ϵ} where $\epsilon > 0$ is an arbitrarily small positive constant.) Refer to section 2.4.2 for the definitions of the problems.

The conclusion for MaxClique follows, of course, from the FGLSS reduction and the first proof system listed above. The conclusion for the Chromatic Number follows from a recent reduction of Fürer [45], which in turn builds on reductions in [71, 66, 23]. (Fürer's work and ours are contemporaneous and thus we view the $N^{\frac{1}{5}}$ hardness result as jointly due to both papers.)

The improvements for the MaxSNP problems are perhaps more significant than the improvements for the MaxClique problem; we see hardness results for MaxSNP problems that are comparable to the factors achieved by known polynomial-time approximation algorithms.

We obtain the first explicit and reasonable nonapproximability factors for Max2SAT, MaxCUT, and minimum Vertex Cover. Recall that the latter is approximable within 2-$o(1)$ [14, 74]. Our results for MaxCUT and Max2SAT show that it is infeasible to find a solution with value which is only a factor of 1.01 from optimal. This may be contrasted with the recent results of [51, 39] which show that solutions which are within 1.14 and 1.075, respectively, of the optimum are obtainable in polynomial time. Thus, even though we do not know if the "pcp approach" allows us to

get the best possible nonapproximability results for these problems, we feel that the current results are not ridiculously far from the known upper bounds.

General framework. We emphasize a general framework for the derivation of strong nonapproximability results for MaxSNP problems which results from our tests and proof systems. We use direct reductions from verifiers to the problems of interest. (This follows and extends [21], prior to which results had used "generic" reductions, which did not take advantage of the nature of the tests performed by the verifier.) In particular, in our case it turns out that the verifier performs only two kinds of tests: (1) verify that $a+b+c=\sigma \pmod 2$; and (2) verify that $a+b_c=\sigma_c$, where a, b, b_0, b_1, c are answer bits obtained from the oracle and the σ's are fixed bits. By constructing local gadgets (i.e., one gadget per random coin toss sequence) to verify each of the verifier's tests, we achieve better nonapproximability results than by using more general reductions. In particular, our work seems to suggest that optimizing for gadgets which "check" the two conditions listed above will lead to reasonably good lower bounds for many MaxSNP problems. In this way, obtaining a nonapproximability result for a particular problem is reduced to the construction of appropriate "gadgets" for "representing" two simple functions.

Techniques. Our main technical contribution is a new error-correcting code which we have called the "long code." This code encodes an n-bit string as a 2^{2^n} bit string which consists of the value of every Boolean function on the n-bit string. It is easy to see that such codes have large Hamming distance. We show that this code is also easily "testable" and "correctable" and derive the new proof systems based on this.

As in all recent constructions of efficient pcp's, our construction also relies on the use of recursive construction of verifiers introduced by Arora and Safra [9]. We have the advantage of being able to use, at the outer level, the recent verifier of Raz [79], which was not available to previous authors. The inner level verifier relies on the use of a "good" encoding scheme. Beginning with [8], constructions of this verifier have used the Hadamard code; in this paper we use instead the long code.

1.2.2. Proofs and approximation: Potential and limits. As the above indicates, nonapproximability results are getting steadily stronger, especially for MaxClique. How far can they go? And, in minimizing amortized free bits, are we on the right track? Are there other ways? The next set of results provides answers to these kinds of questions.

Reversing the connection: Making proofs from gaps. The FGLSS reduction lemma indicates that one route to good nonapproximability results for MaxClique is to show NP $\subseteq \overline{\text{FPCP}}[\log, f]$ for values of f which are as small as possible. We present a "reverse connection" which says that, in a sense, this is the *only* way to proceed. Namely, we "invert" the above FGLSS reduction. Roughly, we show that, for any constant f, the following statements are equivalent:

(1) NP reduces to Gap-MaxClique$_{c,s}$ with gap $c(N)/s(N) = N^{1/(1+f)}$.

(2) NP reduces to $\overline{\text{FPCP}}[\log, f]$.

The (2)⇒(1) direction is the FGLSS reduction; the (1)⇒(2) direction is our reversed connection. (The statement ignores terms of $\epsilon > 0$ which can be arbitrarily small. The proof and a more precise statement are in section 8.) In both cases the reduction is randomized. Furthermore, the statement holds for both Karp and Cook reductions. Also, if (1) holds with a deterministic Karp reduction, then NP $\subseteq \overline{\text{FPCP}}'[\log, f]$, where FPCP$'$ is defined as being the amortized free-bit complexity of proof systems with almost-perfect completeness (i.e., $c = 1 - o(1)$).

In other words, *any* method of proving NP-hardness of MaxClique approximation to a factor of $N^{1/(1+f)}$ implies that NP has proof systems of amortized free-bit complexity f.

We stress both the "qualitative" and "quantitative" aspects of this result. Qualitatively, it provides an answer to the following kind of question: "What do proofs have to do with approximating clique size, and can we not prove non-approximability results without using proof checking?" The result indicates that proofs are inherent, and explains, perhaps, why hardness results avoiding the proof connection have not appeared.

However, at this stage it is the quantitative aspect that interests us more. It says that to get tighter results on MaxClique hardness, we must construct proof systems to minimize the amortized free-bit complexity. So our current efforts (recall that we have the amortized free-bit complexity down to 2, yielding a $N^{\frac{1}{3}}$ hardness for MaxClique) are in the right direction. To prove that, say, MaxClique is hard to approximate within $\sqrt{N}$, our reverse connection says we must construct proof systems with amortized free-bit complexity 1.

Yet the reverse connection does more than guide our choice of parameters. It is also a useful conceptual tool since it allows us to go from graphs to proof systems and vice versa, in the process perhaps gaining some property. As an example we show how all known hardness results for chromatic number can be viewed (with almost no loss in efficiency) as reductions from MaxClique, even though these were essentially hardness results based on proof checking. Other examples demonstrating the usefulness of the equivalence may be found in section 8.4. We believe that further exploring and exploiting this duality is a fruitful avenue to pursue.

A LOWER BOUND ON AMORTIZED FREE BITS. Having shown that the minimization of amortized free bits is unavoidable, we asked ourselves how low we can take them. Our approach here was to look at current techniques and assess their limitations. We stress that this approach makes various assumptions about methods and is intended to show that significantly novel techniques are required to go further. But it does not suggest an *inherent* limitation.

We show that, under the framework used within this and previous papers on this subject, amortized free-bit complexity of 2 seems to be a natural barrier: any proof system in this framework must use $2 - \epsilon$ amortized free bits, where $\epsilon > 0$ as usual can be arbitrarily small. The result, including a definition of what we mean by the "framework," is in section 9. Loosely speaking, it considers proof systems which, among other things, probe two oracles in order to check that one oracle is "close" to a codeword (i.e., a codeword test) and the second oracle encodes a projection of the information encoded in the first oracle (i.e., a projection test).

In retrospect, our lower bounds justify Håstad's two deviations from the current framework; specifically, his relaxation of the codeword test [55] and his relaxation of the projection test [56]. Specifically, Håstad [55, 56] constructed a pcp system (for NP) of amortized free-bit complexity ϵ, $\forall \epsilon > 0$. This was done in two stages/papers. In his first paper [55], Håstad builds on the framework presented in the current work but introduces a relaxed codeword test which is conducted within amortized free-bit complexity ϵ. In his second paper [56], he further modifies the current framework and utilizes a relaxed projection test which is conducted within amortized free-bit complexity ϵ. Our lower bounds justify Håstad's deviations from the intuitive but more stringent forms of the codeword and projection tests.

1.2.3. Properties and transforms of PCP and FPCP. Probabilistic proofs involve a vast arena of complexity parameters: query complexity, free-bit complexity, amortized free-bit complexity, randomness, and proof sizes, to name a few. Some might, at first glance, seem less "natural" than others, yet all are important in applications. A better understanding of the basic properties and relations between these parameters would help move us forward.

We initiate, therefore, a systematic investigation of the properties of pcp complexity classes as a function of the parameter values. Besides providing new results, we take the opportunity to state and prove a few folklore results.

A contribution of this work is to distill and formalize the role of randomized reductions. These transforms provide an elegant and concise way to state connections between PCPs and approximability, or just between different kinds of proof systems, and make it easier to manipulate the many connections that exist to derive new results.

We begin with "triviality results," namely, results which say that certain parameter combinations yield classes that are probably not capable of capturing NP. For simplicity we restrict our attention in this part to classes of languages, not classes of promise problems.

Triviality results. Perhaps the first thing to ask is whether, instead of amortized free-bit complexity, we could work with any of the simpler measures. After all, $\overline{\text{FPCP}}[\log, f]$ contains each of the following classes:

$$\text{PCP}_{1,1/2}[\log, f] \ ; \ \overline{\text{PCP}}[\log, f] \ ; \ \text{FPCP}_{1,1/2}[\log, f] \ .$$

Thus it would suffice to minimize the query complexity to get error 1/2; or the amortized query complexity; or the free-bit complexity to get error 1/2. However, it turns out that these complexities will not enable us to reach our target (of reducing the complexity to almost zero and thus proving that clique is hard to approximate to within a $N^{1-\epsilon}$ factor, for every $\epsilon > 0$). This is because the following classes are all contained in P:

(1) $\text{PCP}_{1,1/2}[\log, 2]$,
(2) $\overline{\text{PCP}}[\log, 1]$,
(3) $\text{FPCP}_{1,1/2}[\log, 1]$.

Thus, we cannot expect to construct pcp systems for NP with either query complexity 2 (this is actually folklore predating our work), or amortized query complexity 1, or free-bit complexity 1. However, it is a feature of amortized free-bit complexity that so far it seems entirely possible that NP reduces to $\overline{\text{FPCP}}[\log, f]$, with f an arbitrarily small constant. Indeed, if we believe (conjecture) that MaxClique is hard to approximate with $N^{1-\epsilon}$ for any $\epsilon > 0$, then such proof systems must exist, by virtue of the equivalence stated above. In fact, it turns out that the amortized query bit parameter is too weak to capture the hardness of the clique function: if MaxClique is hard to approximate to within N^{α}, then the best hardness result obtainable from the amortized query bit parameter would be of the form $N^{\frac{\alpha}{2-\alpha}}$. This is shown by invoking Corollary 10.11 which shows that the amortized query complexity parameter is always one unit larger than the amortized free-bit parameter (and we know that the amortized free-bit parameter captures the hardness of MaxClique tightly).

Other results. We have already mentioned above that strict limitations on various query parameters make PCP very weak. Actually, for every $s < 1$, the classes $\text{PCP}_{1,s}[\log, 2]$ and $\text{FPCP}_{1,s}[\log, 1]$ collapse to P. This means that pcp systems with

perfect completeness are very weak when restricted to either *two queries* or to *free-bit complexity* 1. However, pcp systems with completeness error and the very same query (resp., free-bit) bounds are not so weak. In particular, it is well known that $\mathrm{NP} = \mathrm{PCP}_{c,s}[\log, 2]$ for some $0 < s < c < 1$ (e.g., by using the NP-hardness of approximating Max2SAT). We show that $\mathrm{NP} = \mathrm{FPCP}_{c,s}[\log, 1]$ for some $0 < s < c < 1$ (specifically, $c = 1/2$ and $s = 0.8 \cdot c$). Furthermore, for some smaller $0 < s < c < 1$, the following holds:

$$\mathrm{NP} = \mathrm{FPCP}_{c,s}[\log, 0] \tag{1}$$

(specifically, with $c = \frac{1}{4}$ and $s = \frac{1}{5}$). We find the last assertion quite intriguing. It seems to indicate that one needs to be very careful when making conjectures regarding free-bit complexity. Furthermore, one has to be very careful also when making conjectures regarding amortized free-bit complexity; for example, the result $\mathrm{P} = \overline{\mathrm{PCP}}[\log, 1]$ holds also when one allows nonperfect completeness (in the definition of $\overline{\mathrm{PCP}}[\cdot, \cdot]$) as long as the gap is greater than 2^q per q queries, but an analogous result cannot hold for *two-sided error* amortized free-bit complexity (i.e., $\overline{\mathrm{FPCP}}[\cdot, \cdot]$).

Trying to understand the power of pcp systems with low free-bit complexity, we have waived the bound on the randomness complexity. Recall that in this case pcp systems are able to recognize nondeterministic exponential time (i.e., $\mathrm{NEXPT} = \mathrm{PCP}_{1,1/2}[\mathrm{poly}, \mathrm{poly}]$) [11]. Thus, it may be of interest to indicate that for every $s < 1$,

$$\mathrm{FPCP}_{1,s}[\mathrm{poly}, 0] \subseteq \mathrm{coNP}, \tag{2}$$

$$\mathrm{FPCP}_{1,s}[\mathrm{poly}, 1] \subseteq \mathrm{PSPACE}. \tag{3}$$

It seems that $\mathrm{FPCP}_{1,1/2}[\mathrm{poly}, 0]$ is not contained in BPP, since quadratic nonresiduosity and graph nonisomorphism belong to the former class. (Specifically, the interactive proofs of [53] and [52] can be viewed as a pcp system with polynomial randomness, query complexity 1, and free-bit complexity 0.) Thus, it seems that the obvious observation $\mathrm{PCP}_{1,s}[\mathrm{poly}, 1] \subseteq \mathrm{AM}$ (for every $s < 1$, where AM stands for one round Arthur–Merlin games) would also be hard to improve upon.

Transformations between proof systems. We provide various useful transformations of pcp systems. These transformations are analogous to transformations that can be applied to graphs with respect to the MaxClique problem. In view of the relation (mentioned above), between FPCP and the gap-clique promise problem, this analogy is hardly surprising.

One type of transformation amplifies the gap (i.e., the ratio between completeness and soundness bounds) of the proof system while preserving its amortized free-bit complexity and incurring a relatively small additional cost in the randomness complexity. Specifically, using a randomized reduction we can transform $\mathrm{FPCP}_{1,1/2}[\log, f]$ into $\mathrm{FPCP}_{1,2^{-k}}[\log + k, k \cdot f]$ (ignoring multiplicative factors of $1 + \epsilon$ for arbitrarily small $\epsilon > 0$). This transformation is analogous to the well-known transformation of Berman and Schnitger [25]. Alternatively, using a known deterministic amplification method based on [2, 70], one can transform $\mathrm{FPCP}_{1,1/2}[\log, f]$ into $\mathrm{FPCP}_{1,2^{-k}}[\log + 2k, k \cdot f]$. Both alternatives are important ingredients in transforming pcp results into clique inapproximability results via the FGLSS method.

A second type of transformation moves the location of the gap (or, equivalently, the completeness parameter). The gap itself is preserved by the transformation but moving it is related to changing the free-bit complexity (and thus the amortized free-bit complexity is not preserved). Moving the gap "up" requires increasing the

free-bit complexity, whereas moving the gap "down" allows us to decrease the free-bit complexity. For example, we randomly reduce $\mathrm{FPCP}_{c,s}[\log, f]$ to $\mathrm{FPCP}_{1,s\cdot\log}[\log, f + \log(1/c) + \log\log]$. On the other hand, for every $k \leq f$, we (deterministically) reduce $\mathrm{FPCP}_{c,s}[\log, f]$ into $\mathrm{FPCP}_{\frac{c}{2^k},\frac{s}{2^k}}[\log, f-k]$, provided that the original system has at least 2^k accepting configurations per each possible sequence of coin tosses. (This condition is satisfied in many natural pcp systems, even for $k = f$.)

1.3. History. Early work in nonapproximability includes that of Garey and Johnson [47] showing that it is NP-hard to approximate the chromatic factor within a factor less than 2. The indication of higher factors, and results for other problems, had to wait for the PCP connection.

Interactive proofs were introduced by Goldwasser, Micali, and Rackoff [53] and Babai [10]. Ben-Or et al. [24] extended these ideas to define a notion of multiprover interactive proofs. Fortnow, Rompel, and Sipser [44] showed that the class MIP of languages possessing multiprover interactive proofs equals the class of languages which have (using today's terms) probabilistically checkable proofs (of unrestricted, and thus polynomial, randomness, and query complexity).

The first indication of the power of interactive proof systems was given in [52], where it was shown that interactive proofs exist for graph nonisomorphism (whereas this language is not known to be in NP). However, the real breakthrough came with the result of Lund et al. [72] who used algebraic methods to show that all coNP languages (and actually, all languages in $\mathrm{P}^{\#P}$) have interactive proof systems. These techniques were used by Shamir [81] to show that IP = PSPACE.

A central result that enabled the connection to hardness of approximation is that of Babai, Fortnow, and Lund [11]. They showed that the class MIP equals the class NEXP (i.e., languages recognizable in nondeterministic exponential time). The latter result has been "scaled down" to the NP-level by two independent groups of researchers. Babai et al. [12] showed that if the input is encoded using a special error-correcting code (for which encoding and decoding can be performed in polynomial time), then NP has transparent proof systems (i.e., it is possible to verify the correctness of the proof in polylogarithmic time). Feige et al. [40] showed that NP has probabilistically checkable proofs of polylogarithmic randomness and query complexity; namely, $\mathrm{NP} \subseteq \mathrm{PCP}_{1,1/2}[r, q]$, where $r(n) = q(n) = O(\log n \cdot \log\log n)$.

A hardness of approximation result based on interactive proofs was first proved by Condon [31]. The breakthrough *PCP connection (to approximation)* was made by Feige et al. [40]. They showed that $\mathrm{NP} \subseteq \mathrm{PCP}_{1,s}[r, q]$ implies that approximating the maximum clique in a $2^{r(n)+q(n)}$-vertices graph to within a $1/s(n)$ factor is infeasible (i.e., not doable in polynomial-time), provided that NP is not in $\mathrm{Dtime}(2^{O(r+q)})$. (Here n is the length of the input x to the pcp verifier.) Combined with the above-mentioned results, they obtained the first in a sequence of strong nonapproximability results for MaxClique: a nonapproximability factor of $2^{\log^{1-\epsilon} N}$, $\forall \epsilon > 0$, assuming NP does not have quasi-polynomial time algorithms.

After the work of [40] the field took off in two major directions. One was to extend the interactive proof approach to prove the nonapproximability of other optimization problems. Direct reductions from proofs were used to show the hardness of quadratic programming [22, 43], Max3SAT [8], set cover [71], and other problems [16]. The earlier work of Papadimitriou and Yannakakis, introducing the class MaxSNP [76], now came into play; by reduction from Max3SAT it implied hardness of approximation for any MaxSNP-hard problem. Also, reductions from MaxClique lead to hardness results for the Chromatic Number [71] and other problems [86].

The other direction was to increase factors, and reduce assumptions, for existing hardness of approximation results. This involves improving the efficiency of the underlying proof systems and/or the efficiency of the reductions.

The first stage of this enterprise started with the work of Arora and Safra [9]. They showed that NP $\subseteq$ PCP$_{1,1/2}[\log, o(\log)]$. This provided the first strong NP-hardness result for MaxClique (specifically, a hardness factor of $2^{\sqrt{\log N}}$). This work introduced the idea of recursive proof checking, which turned out to play a fundamental role in all subsequent developments. Interestingly, the idea of encoding inputs in an error-correcting form (as suggested in [12]) is essential to make "recursion" work. Arora et al. [8] reduced the query complexity of pcp systems for NP to a constant while preserving the logarithmic randomness complexity; namely, they showed that NP = PCP$_{1,1/2}[\log, O(1)]$. This immediately implied the NP-hardness of approximating MaxClique within N^{ϵ}, for some $\epsilon > 0$. Furthermore, it also implied that Max3SAT is NP-hard to approximate to within some constant factor [8] and so is any MaxSNP-hard problem [76].

The second stage of this enterprise started with the work of Bellare et al. [21]. The goal was to improve (increase) the constant ϵ in the exponent of the hardness of approximation factor for MaxClique and also to improve the constant values of the hardness factors in the MaxSNP hardness results. They presented new proof systems minimizing query complexity and exploited a slightly improved version of the FGLSS reduction due to [25, 86] to get a $N^{1/30}$ hardness of approximation factor for MaxClique. Feige and Kilian [41], however, observed that one should work with free bits, and noted that the free-bit complexity of the system of [21] was 14, yielding a $N^{1/15}$ hardness factor. Bellare and Sudan then suggested the notion of amortized free bits [23]. They constructed proof systems achieving amortized free-bit complexity 3 and thus obtained a $N^{\frac{1}{4}}$ hardness for MaxClique assuming NP $\not\subseteq$ co$\widetilde{\text{RP}}$.

Detailed histories for specific topics are given in sections 2.2.3 and 2.4.3.

1.4. Related work. Following the presentation of our results, Arora has also investigated the limitations of proof checking techniques in proving nonapproximability results [6]. As in our free-bit lower bound result, he tries to assess the limitations of current techniques by making some assumptions about these techniques and then showing a lower bound. His focus is on the reductions, which he assumes are "code like." In this setting he can show that one should not expect to prove nonapproximability of MaxClique within $N^{1/2}$. (The assumptions made by us and by Arora do not seem to be comparable; neither implies the other. In retrospect, both sets of assumptions could be bypassed as done by Håstad [55, 56].)

1.5. Subsequent work. Our prophecy that the PCP approach is leading to tight nonapproximability results is in the process of materializing (see Figure 3). By now, tight results are known for central problems such as Min Set Cover (cf. [71, 21, 38]), MaxClique (cf. [55, 56]), Min Coloring [42], and Max3SAT (cf. [57]). The latter results were obtained subsequent to, and while building on the current work.

Amortized free bits and MaxClique. The most intriguing problem left open by our work has been resolved by Håstad [55, 56]. He proved our conjecture (cf. [20]) by which, for every $\epsilon > 0$, it is the case that NP $\subseteq \overline{\text{FPCP}}[\log, \epsilon]$. The long code, introduced in this work, plays a pivotal role in Håstad's work. He also uses the idea of folding (introduced here). Applying the FGLSS reduction to the new proof system, Håstad obtains a tight result for MaxClique by showing that for every $\epsilon > 0$, assuming

Problem	EASY to Approx. Factor		HARD to Approx. Factor			Tight?
	Factor	Due to	Ours	Subsequent	Assumption	
Max3SAT	$1+\frac{1}{7}+\epsilon$	[64]	$1+\frac{1}{26}$	$1+\frac{1}{7}-\epsilon$ [57]	P $\neq$ NP	Yes
MaxE3SAT	$1+\frac{1}{7}$	folklore	$1+\frac{1}{26}$	$1+\frac{1}{7}-\epsilon$ [57]	P $\neq$ NP	Yes
Max2SAT	1.075	[51, 39]	1.013	1.047 [57]	P $\neq$ NP	No
Max$\oplus$SAT	2	folklore	$1+\frac{1}{7}-\epsilon$	$2-\epsilon$ [57]	P $\neq$ NP	Yes
MaxCUT	1.139	[51]	1.014	1.062 [57]	P $\neq$ NP	No
MinVC	$2-o(1)$	[14, 74]	$1+\frac{1}{15}$	$1+\frac{1}{6}-\epsilon$ [57]	P $\neq$ NP	No
MaxClique	$N^{1-o(1)}$	[28]	$N^{\frac{1}{3}-\epsilon}$	$N^{1-\epsilon}$ [56]	coRP $\neq$ NP	Yes
			$N^{\frac{1}{4}-\epsilon}$	$N^{\frac{1}{2}-\epsilon}$ [56]	P $\neq$ NP	No
Chrom. No.	$N^{1-o(1)}$	[28]	$N^{\frac{1}{5}-\epsilon}$	$N^{1-\epsilon}$ [42]	coRP $\neq$ NP	Yes

FIG. 3. *State of the art regarding easy and hard approximation factors* (*updated July* 1997). *Here* $\epsilon > 0$ *is an arbitrarily small constant.*

NP $\neq$ coRP, there is no polynomial-time algorithm to approximate MaxClique within a factor of $N^{1-\epsilon}$.

IMPROVED 3-QUERY PROOFS AND MAXSNP. Another challenge, one we even did not dare state, was achieved as well. Håstad [57] has recently obtained optimal nonapproximability results to MaxSNP problems such as Max3SAT. Furthermore, he has improved over all our nonapproximability results for MaxSNP problems, obtaining nonapproximability factors of 22/21 and 17/16 for Max2SAT and MaxCUT, respectively. Underlying these results is a new proof system for NP which yields NP $\subseteq$ PCP$_{1-\epsilon,0.5}[\log, 3]$, for any $\epsilon > 0$. In addition, Håstad [57] shows that NP is contained in PCP$_{1,0.75+\epsilon}[\log, 3]$ (and it follows that NP $\subseteq$ PCP$_{1,0.5}[\log, 9]$). The long code plays a pivotal role in all of these proof systems.

IMPROVED 2-FREE-BITS PROOFS AND MINVC. The above-mentioned proof system of Håstad [57] uses two (nonamortized) free bits, and so NP $\subseteq$ FPCP$_{1-\epsilon,0.5}[\log, 2]$, for every $\epsilon > 0$. This sets the nonapproximability bound for Min Vertex Cover at $\frac{7}{6}-\epsilon$.

CHROMATIC NUMBER. Feige and Kilian [42] have introduced a new approach to showing hardness of approximability of ChromNum, based on a new measure of proof checking complexity called the covering complexity. By modifying our proof systems so as to preserve the amortized free-bit complexity and achieve low covering complexity, they proved that approximating ChromNum within $N^{\frac{1}{3}}$ is hard unless NP = coRP. They were able to similarly modify Håstad's proof systems [55, 56] and thereby improve the hard factor to $N^{1-\epsilon}$, for any $\epsilon > 0$.

GADGETS. Another research direction, suggested in early versions of this work [20], was taken on by Trevisan et. al. [84] who initiated a systematic study of the construction of gadgets. In particular, they showed that the gadgets we have used in our reductions to the MaxSAT problems were optimal, and constructed better (and actually optimal) gadgets for reduction to MaxCUT.

WEIGHTS. An important issue neglected in our treatment of MaxSNP problems is that of weights. For example, in our MaxSAT problems we have allowed the same clause to appear many times in the formula, which can be considered as allowing "small" weights. Certainly, one may want nonapproximability results for the unweighted case (where one does not allow multiple occurrences of the same clause). This issue is treated in a subsequent paper by Crescenzi, Silvestri, and Trevisan [34]. Essentially, they show that the unweighted cases of all problems considered in our paper are as hard as the weighted cases.

1.6. Directions for further research. Although the most intriguing open problems suggested in previous versions of this work [20] have been resolved, some very interesting problems remain. We mention a few.

2-FREE-BITS PROOFS AND MINVC. As we show, NP $\subseteq$ FPCP$_{c,s}[\log, f]$ implies that approximating Min Vertex Cover up to a $\frac{2^f-s}{2^f-c}$ factor is NP-hard. This motivates us to ask whether the following, increasingly stronger, conjectures hold.

(1) NP $\subseteq$ FPCP$_{1-\epsilon,\epsilon}[\log, 2]$ (or even NP $\subseteq$ FPCP$_{1,\epsilon}[\log, 2]$) for every $\epsilon > 0$. This would imply a hardness factor of $\frac{4}{3} - \epsilon$ for MinVC.

(2) For $f \stackrel{\text{def}}{=} \log_2 3$, NP $\subseteq$ FPCP$_{1-\epsilon,\epsilon}[\log, f]$ (or even NP $\subseteq$ FPCP$_{1,\epsilon}[\log, f]$) for every $\epsilon > 0$. This would imply a hardness factor of $\frac{3}{2} - \epsilon$.

(3) NP $\subseteq$ FPCP$_{1-\epsilon,\epsilon}[\log, 1]$ for every $\epsilon > 0$. This would imply a hardness factor of $2 - \epsilon$.

Recall that FPCP$_{1,s}[\log, 1] \subseteq$ P, for every $s < 1$, whereas NP $\subseteq$ FPCP$_{1-\epsilon,0.5}[\log, 2]$ [57]. It will be even interesting (though of no application for MinVC) to know whether NP $\subseteq$ FPCP$_{1,0.5+\epsilon}[\log, 2]$, for every $\epsilon > 0$.

PERFECT VERSUS IMPERFECT COMPLETENESS. Håstad's work [57] is indeed the trigger for the last question and similarly, we wonder whether NP $\subseteq$ PCP$_{1,0.5+\epsilon}[\log, 3]$. Nonperfect completeness seems to be useful in [57], but it is to be seen if this usage is inherent. Similar issues arise with respect to some results in the current work (e.g., see our transformations for increasing acceptance probability of proof systems).

DERANDOMIZATION. We know that $\overline{\text{FPCP}}[\log, f]$ is *randomly* reducible to

$$\text{FPCP}_{1,2^{-k}}[\log +(1+\epsilon)k, (1+\epsilon)k \cdot f] \,.$$

On the other hand, the former class is contained in (i.e., is *deterministically* reduced to) the class FPCP$_{1,2^{-k}}[\log +(2+\epsilon)k, (1+\epsilon)k \cdot f]$, for arbitrarily small $\epsilon > 0$. Can one obtain the best of both worlds, namely, a deterministic reduction of $\overline{\text{FPCP}}[\log, f]$ to, say, FPCP$_{1,2^{-k}}[\log +(1+\epsilon)k, (1+\epsilon)k \cdot f]$, for arbitrarily small $\epsilon > 0$? An affirmative answer will allow us to infer from NP $\subseteq \overline{\text{FPCP}}[\log, f]$ that approximating Max Clique to within an $N^{\frac{1}{1+f+\epsilon}}$ factor is NP-hard (rather than NP-hard under randomized reductions).

One ingredient of our method for reversing the FGLSS reduction is the randomized reduction of the class FPCP$_{c,s}[\log, f]$ to the class FPCP$_{1,\frac{\log}{c}.s}[\log, f + \log(1/c) + \log\log]$. (This statement is proved using the ideas in section 11. An alternative exposition, making use of a randomized graph-layering process, is given in section 8.) Anyhow, randomness plays an essential role in obtaining a pcp system with perfect completeness.[2] The question is whether the class FPCP$_{c,s}[\log, f]$ is contained in the class FPCP$_{1,\frac{\log}{c}.s}[\log, f + \log(1/c) + \log\log]$ (rather than being randomly reducible to it).

1.7. Organization. This introduction is followed by a section that contains definitions as well as detailed histories. The main content of the paper is divided into three parts:

Part I—*New proof systems and nonapproximability results*—consisting of sections 3 to 7, contains the material discussed in section 1.2.1. See overview in section 3.1.

[2]This makes our results more elegant, but actually, as indicated in section 8, we could have settled for "almost perfect" completeness which suffices for presenting an inverse of the "FGLSS reduction."

Part II—*Proofs and approximation: Potential and limitations*—consisting of sections 8—9, contains the material discussed in section 1.2.2. Specifically, section 8 contain the "reverse reduction" of clique hardness to PCP, and section 9 contains lower bounds on the free-bit complexity of certain tasks.

Part III—*PCP: Properties and transformations*—consisting of sections 10 and 11, contains the material discussed in section 1.2.3. Specifically, section 10 studies the expressive power of PCP systems with certain parameters, and section 11 contains transformations among PCP classes.

2. Definitions and histories.

2.1. General notation and definitions. For a set S, we denote by $e \stackrel{R}{\leftarrow} S$ the operation of selecting e uniformly in S. For integer n let $[n] = \{1, \ldots, n\}$. A graph always means an undirected graph with no self-loops, unless otherwise indicated. We let $\|G\|$ denote the number of vertices in graph $G = (V, E)$.

A probabilistic machine K has one or more inputs, denoted $x_1, x_2, \ldots$. It tosses coins to create a random string R, usually of some length $r(\cdot)$ which is a function of the (lengths of the) inputs. We let $K(x_1, x_2, \ldots; R)$ denote the output of K when it uses the random string, denoted R. Typically we are interested in the probability space associated with a random choice of R.

A function is *admissible* if it is polynomially bounded and polynomial-time computable. We will ask that all functions measuring complexity (e.g., the query complexity $q = q(n)$) be admissible.

In defining complexity classes we will consider promise problems rather than languages. (This convention is adopted since approximation problems are easily cast as promise problems.) Following Even, Selman, and Yacobi [36], a promise problem is a pair of disjoint sets (A, B), the first being the set of "positive" instances and the second the set of "negative" instances. A language L is identified with the promise problem $(L, \overline{L})$.

2.2. Proof systems.

2.2.1. Basic setting. A verifier is a probabilistic machine V taking one or more inputs, and it is also allowed access to one or more oracles. Let x denote the sequence of all inputs to V and let n denote its length. During the course of its computation on coins R and input x, the verifier makes queries of its oracles. Its final decision to accept or reject is a function $\mathsf{DEC}_V(x, a; R)$ of x, R and the sequence a of all the bits obtained from the oracle in the computation. Contrary to standard terminology, acceptance in this paper will correspond to outputting 0 and rejection to outputting 1. This is done since, in most technical discussions, our focus is on the reject event.

Oracles are formally functions, with the context specifying for each the domain and range. Sometimes, however, an algorithm will be given a string s as an oracle. (Giving a verifier s as an oracle models a "written proof" model in which someone has "written" s somewhere and the verifier wants to check it.) This is to be interpreted in the natural way; namely, the oracle takes an index i and returns the ith bit of s. Let π denote the sequence (tuple) of all proof oracles supplied to the verifier V. Now for verifier V, examining the proofs π and having input x, we let

$$\mathtt{ACC}\left[\, V^{\pi}(x) \,\right] \;=\; \Pr_R \left[V^{\pi}(x; R) = 0 \right]$$

denote the probability that V accepts when its random string is R. We then let

$$\mathtt{ACC}\left[\, V(x) \,\right] \;=\; \max_{\pi} \; \mathtt{ACC}\left[\, V^{\pi}(x) \,\right].$$

This is the maximum accepting probability over all possible choices of proof sequences π. (The domain from which the proofs are chosen depends, as mentioned above, on the context.)

Let $\mathsf{pattern}_V(x;R)$ be the set of all sequences a such that $\mathsf{DEC}_V(x,a;R)=0$. (That is, all sequences of oracle answers leading to acceptance). A *generator* for V is a poly(n)-time computable function G such that $G(x,R)=\mathsf{pattern}_V(x;R)$ for all x,R. (That is, it can efficiently generate the set of accepted patterns.)

2.2.2. Parameters. We are interested in a host of parameters that capture various complexity measures of the proof checking process. They are all functions of the length n of the input x given to the verifier V. In the following, σ denotes the concatenation of all the proof strings given to the verifier. Also recall that we are interested in proof systems for promise problems (A,B) rather than just for languages.

$\mathsf{coins}=$ Number of coins tossed by verifier. Typically denoted r.

$\mathsf{pflen}=$ Length of the proof provided to the verifier. Typically denoted l.

$c=$ Completeness probability. Namely,

$$\min\{\,\mathtt{ACC}[\,V(x)\,]\;:\;x\in A \text{ and } |x|=n\,\}\,.$$

$s=$ Soundness probability. Namely,

$$\max\{\,\mathtt{ACC}[\,V(x)\,]\;:\;x\in B \text{ and } |x|=n\,\}\,.$$

$g=$ Gap. Namely c/s.

Now we move to various measures of the "information" conveyed by the oracle to the verifier. For simplicity we consider here only oracles that return a single bit on each query; that is, they correspond to strings, or "written proofs."

$\mathsf{query}=$ The query complexity on input x is the maximum, over all possible random strings R of V, of the number of bits of σ accessed by V on input x. The query complexity of the system $q=q(n)$ is the maximum of this over all inputs $x\in A\cup B$ of length n.

$\mathsf{query}_{\mathrm{av}}=$ The average query bit complexity on input x is the average, over R, of the number of bits of the proof σ accessed by V on input x and coins R. The average query complexity of the system is the maximum of this over all $x\in A\cup B$ of length n. Typically denoted q_{av}.

$\overline{\mathsf{query}}=$ V is said to have amortized query bit complexity $\bar{q}$ if $q/\lg(g)\le\bar{q}$, where q is the query bit complexity and g is the gap, and, furthermore, q is at most logarithmic in n.

$\mathsf{free}=$ The free-bit complexity of V is f if there is a generator G such that $|G(x,R)|\le 2^f$ for all R and all $x\in A\cup B$ of length n.

$\mathsf{free}_{\mathrm{av}}=$ The average free-bit complexity of V is f_{av} if there is a generator G such that $\mathbf{E}_R[|G(x,R)|]\le 2^{f_{\mathrm{av}}}$ for all $x\in A\cup B$ of length n.

$\overline{\mathsf{free}}=$ V is said to have amortized free-bit complexity $\bar{f}$ if $f/\lg(g)\le\bar{f}$, where f is the free-bit complexity and g is the gap.

Notice that amortized query complexity is restricted to be at most logarithmic. We don't need to explicitly make this restriction for the amortized free-bit complexity since it is a consequence of the efficient generation condition.

In case the completeness parameter equals 1 (i.e., $c=1$), we say that the system is of *perfect completeness*. In case the completeness parameter, c, satisfies $c(n)=1-o(1)$, we say that the system is of *almost-perfect completeness*.

The consideration of combinations of all these parameters gives rise to a potentially vast number of different complexity classes. We will use a generic notation in which the parameter values are specified by name, except that, optionally, the completeness and soundness can, if they appear, do so as subscripts. Thus for example we have:

$$\mathrm{PCP}_{c,s}[\,\mathsf{coins} = r\,;\ \mathsf{query} = q\,;\ \mathsf{pflen} = 2^r\,;\ \mathsf{free} = f\ldots]\,.$$

However most often we'll work with the following abbreviations:

$$\begin{aligned}
\mathrm{PCP}_{c,s}[r,q] &\stackrel{\text{def}}{=} \mathrm{PCP}_{c,s}[\,\mathsf{coins} = r\,;\ \mathsf{query} = q\,],\\
\overline{\mathrm{PCP}}_c[\,r,\,q\,] &\stackrel{\text{def}}{=} \mathrm{PCP}_{c,\cdot}[\,\mathsf{coins} = r\,;\ \overline{\mathsf{query}} = q\,],\\
\mathrm{FPCP}_{c,s}[r,f] &\stackrel{\text{def}}{=} \mathrm{PCP}_{c,s}[\,\mathsf{coins} = r\,;\ \mathsf{free} = f\,],\\
\mathrm{FPCP}_{c,s}[r,f,l] &\stackrel{\text{def}}{=} \mathrm{PCP}_{c,s}[\,\mathsf{coins} = r\,;\ \mathsf{free} = f\,;\ \mathsf{pflen} = l\,],\\
\overline{\mathrm{FPCP}}_c[\,r,\,f\,] &\stackrel{\text{def}}{=} \mathrm{PCP}_{c,\cdot}[\,\mathsf{coins} = r\,;\ \overline{\mathsf{free}} = f\,]\,.
\end{aligned}$$

We stress that in the definitions of the amortized classes, $\overline{\mathrm{PCP}}_c[\,r,\,q\,]$ and $\overline{\mathrm{FPCP}}_c[\,r,\,f\,]$, we refer to the completeness parameter c (but not to the soundness parameter). In case $c = 1$, we may omit this parameter and shorthand the amortized classes of perfect completeness by $\overline{\mathrm{PCP}}[\,r,\,q\,]$ and $\overline{\mathrm{FPCP}}[\,r,\,f\,]$, respectively. Namely,

$$\begin{aligned}
\overline{\mathrm{PCP}}[\,r,\,q\,] &\stackrel{\text{def}}{=} \overline{\mathrm{PCP}}_1[\,r,\,q\,],\\
\overline{\mathrm{FPCP}}[\,r,\,f\,] &\stackrel{\text{def}}{=} \overline{\mathrm{FPCP}}_1[\,r,\,f\,].
\end{aligned}$$

2.2.3. History of proof systems.

MODELS AND PARAMETERS. The model underlying what are now known as "probabilistically checkable proofs" is the "oracle model" of Fortnow, Rompel, and Sipser [44], introduced as an equivalent version (with respect to language recognition power) of the multiprover model of Ben-Or et al. [24]. Interestingly, as shown by [12, 40], this framework can be applied in a meaningful manner also to languages in NP. These works provide the verifier V with a "written" proof, modeled as an oracle to which V provides the "address" of a bit position in the proof string and is returned the corresponding bit of the proof. Babai et al. [12] suggested a model in which the inputs are also given as oracles encoded in a special (polynomial-time computable and decodable) error-correcting code and the verifier works in polylogarithmic time. Here we follow the model of Feige et al. [40], where the verifier is probabilistic polynomial time (as usual) and one considers finer complexity measures such as the query and randomness complexity. The FGLSS reduction (cf. [40]), stated in terms of the query complexity (number of binary queries), randomness complexity, and error probability of the proof system, has focused attention on the above model and these parameters. The class $\mathrm{PCP}_{1,1/2}[r,q]$ was made explicit by [9].

The parameterization was expanded by [21] to explicitly consider the answer size (the oracle was allowed to return more than one bit at a time) and query size. Their notation included five parameters: randomness, number of queries, size of each query, size of each answer, and error probability. They also similarly parameterized (single round) multiprover proofs, drawing attention to the analogue with pcp. This served to focus attention on the roles of various parameters, both in reductions and in

Due to	q	q_{av}
[8]	some constant	some constant
[21]	36	29
[41]	32	24
This paper	11	10.9

FIG. 4. *Worst case* (q) *and average* (q_{av}) *number of queries needed to get soundness error* 1/2 *with logarithmic randomness and perfect completeness; that is, results of the form of Eq.* (4).

constructions. Also, they introduced the consideration of average query complexity, the first in a sequence of parameter changes toward doing better for clique.

Free bits are implicit in [41] and formalized in [23]. Amortized free bits are introduced in [23] but formalized a little better here.

Proof sizes were considered in [12, 78]. We consider them here for a different reason—they play an important role in that the randomized FGLSS reduction [25, 86] depends, actually, on this parameter (rather than on the randomness complexity).

The discussion of previous proof systems is coupled with the discussion of MaxClique in section 2.4.3. We conclude the current section by discussing two somewhat related topics: query minimization and constant-prover proof systems.

QUERY COMPLEXITY MINIMIZATION. One seeks results of the form

$$\text{NP} \;=\; \text{PCP}_{1,1/2}[\,\mathsf{coins} = \log\,;\,\mathsf{query} = q\,;\,\mathsf{query}_{\text{av}} = q_{\text{av}}\,]\,. \tag{4}$$

This was originally done for deriving NP-hardness results for the approximation of MaxClique, but subsequent work has indicated that other parameters actually govern this application. Still, the query complexity of a proof system remains a most natural measure, and it is an intriguing question as to how many bits of a proof you need to look at to detect an error with a given probability. Specifically, we consider the question of determining the smallest values of q, q_{av} for which Eq. (4) holds.

The fundamental result of [8] states that q, q_{av} may be constants (independent of the input length). Reductions in the values of these constants were obtained since then and are depicted in Figure 4. See section 6 for our results.

ROLE OF CONSTANT-PROVER PROOFS IN PCP: PERSPECTIVE. Constant-prover proofs have been instrumental in the derivation of nonapproximability results in several ways. One of these is that they are a good starting point for reductions—examples of such are reductions of two-prover proofs to quadratic programming [22, 43] and set cover [71]. However, it is a different aspect of constant-prover proofs that is of direct concern to us. This aspect is the use of constant-prover proof systems as the penultimate step of the recursion, and begins with [8]. It is instrumental in getting PCP systems with only a constant number of queries. Their construction requires that these proof systems have low complexity: error which is any constant and randomness and answer sizes that are preferably logarithmic. The number of provers and the randomness and query complexity determine the quality of many nonapproximability results (e.g., polylogarithmic rather than logarithmic complexities translate into nonapproximability results using assumptions about quasi-polynomial time classes rather than polynomial-time classes). The available constant-prover proof systems appear in Figure 5 and are discussed below. Throughout this discussion we consider proof systems obtaining an *arbitrary small constant* error probability, denoted ϵ.

The two-prover proofs of Lapidot and Shamir [67] and Feige and Lovász [43] have polylogarithmic randomness and answer sizes. Arora et al. [8] used a modification of

Due to	Provers	Coins	Answer size	Canonical?	Can be made canonical?
[67, 43]	2	polylog	polylog	No	Yes [23]
[8]	poly(ϵ^{-1})	log	polylog	No	?
[21]	4	log	polyloglog	No	?
[82]	3	log	$O(1)$	No	?
[41]	2	log	$O(1)$	No	With 3 provers [23]
[79]	2	log	$O(1)$	Yes	(NA)

FIG. 5. *Constant-prover proof systems achieving error which is a fixed, but arbitrarily small, constant* ϵ.

these proofs, so to reduce randomness complexity, in the process increasing the number of provers to a constant much larger than 2. Later constructions of few-prover proofs of [21, 82, 41] lead to better nonapproximability results.

Bellare and Sudan [23] identified some extra features of constant-prover proofs whose presence they showed could be exploited to further increase the nonapproximability factors. These features are captured in their definition of canonical verifiers (cf. section 3.4). But the proof systems of [41] that had worked above no longer sufficed—they are not canonical. So instead, [23] used (a slight modification of) the proofs of [67, 43], thereby incurring polylogarithmic randomness and answer sizes, and so that the assumptions in their nonapproximability results pertain to quasi-polynomial-time classes. (Alternatively they modify the system of [41] to a canonical three-prover system, but then incur a decrease in the nonapproximability factors due to having more provers).

A breakthrough result in this area is Raz's parallel repetition theorem which implies the existence of a two-provers proof system with logarithmic randomness and constant answer size [79]. Furthermore, this proof system is canonical.

2.3. Reductions between problems and classes. We will consider reductions between promise problems. A deterministic Karp reduction from (A_1, B_1) to (A_2, B_2) is a polynomial-time function T which for all x satisfies the following: if $x \in A_1$, then $T(x) \in A_2$, and if $x \in B_1$, then $T(x) \in B_2$. A randomized Karp reduction from (A_1, B_1) to (A_2, B_2) is a probabilistic, polynomial-time process T which takes two arguments, an input x and a security parameter k; the latter written in unary. The transformation is required to have the property that

$$\begin{aligned} x \in A_1 \quad &\Longrightarrow \quad p_1(x,k) \stackrel{\text{def}}{=} \Pr\left[\, T(x,1^k) \in A_2 \,\right] \geq 1 - 2^{-k}, \\ x \in B_1 \quad &\Longrightarrow \quad p_2(x,k) \stackrel{\text{def}}{=} \Pr\left[\, T(x,1^k) \in B_2 \,\right] \geq 1 - 2^{-k}\,. \end{aligned}$$

The probability is over the coin tosses of T. We say the reduction has perfect completeness if $p_1 = 1$ and perfect soundness if $p_2 = 1$. Notice a deterministic reduction corresponds to a randomized reduction in which $p_1 = p_2 = 1$. We write $(A_1, B_1) \leq_R^K (A_2, B_2)$ if there is a randomized Karp reduction from (A_1, B_1) to (A_2, B_2). If the reduction is deterministic we omit the subscript of "R" or, sometimes for emphasis, replace it by a subscript of "D."

An example is the randomized FGLSS transformation [40, 25, 86]. Here (A_1, B_1) is typically an NP-complete language L, and (A_2, B_2) is $\mathsf{Gap}\text{-MaxClique}_{c,s}$ for some c, s which are determined by the transformation. (See section 2.4 for a definition of the latter.) This transformation has perfect soundness, while, on the other hand, it is possible to get $p_1 = 1 - 2^{-\operatorname{poly}(k)}$.

Similarly, one can define (randomized) Cook reductions. The notation for these reductions is $\leq_R^C$.

Let C be a complexity class (e.g., NP). We say that C reduces to (A_2, B_2) if for every (A_1, B_1) in C it is the case that (A_1, B_1) reduces to (A_2, B_2). An example is to say that NP reduces to Gap-MaxClique$_{c,s}$. We say that C_1 reduces to C_2, where C_1 and C_2 are complexity classes, if for every (A_1, B_1) in C_1 there is an (A_2, B_2) in C_2 such that (A_1, B_1) reduces to (A_2, B_2). An example is to say that NP reduces to $\overline{\mathrm{FPCP}}[\log, f]$. The notation of $\leq_R^K$ or $\leq_R^C$ extends to these cases as well.

Notice that our definition of reducibility ensures that this relation is transitive.

For simplicity we sometimes view a randomized reduction T as a function only of x, and write $T(x)$. In such a case it is to be understood that the security parameter has been set to some convenient value, such as $k = 2$.

HISTORICAL NOTE. We've followed the common tradition regarding the names of polynomial-time reductions: many-to-one reductions are called Karp reductions whereas (polynomial-time) Turing reductions are called Cook reductions. This terminology is somewhat unfair toward Levin whose work on NP-completeness [69] was independent of those of Cook [32] and Karp [65]. Actually, the reductions considered by Levin are more restricted as they also efficiently transform the corresponding NP-witnesses (this is an artifact of Levin's desire to treat search problems rather than decision problems). In fact, such reductions (not surprisingly termed Levin reductions) are essential for results such as Corollary 8.15. (Yet this is the only example in the current paper.)

2.4. Approximation problems and quality. We discuss optimization problems, approximation algorithms for them, and how hardness is shown via the "production of hard gaps." We then list all the problems considered in this paper. A continuously updated compendium of NP-optimization problems is available in [33].

2.4.1. Optimization problems, approximation, and gaps. An *optimization problem* $\Phi = (S, g, \|\cdot\|, \|\cdot\|^*, \mathrm{opt})$ is specified by

- a function S associating to any *instance* w a *solution set* $S(w) \neq \emptyset$.
- an *objective function* g associating to any instance w and solution $y \in S(w)$ a nonnegative real number $g(w, y)$. This number is sometimes called the *value* of solution y.
- two *norm* functions $\|\cdot\|, \|\cdot\|^*$, the first admissible, the second polynomial-time computable, each associating to any instance w a nonnegative real number; their roles will be explained later.
- an indication $\mathrm{opt} \in \{\min, \max\}$ of the *type* of optimization, whether maximization or minimization.

The task, given w, is to either maximize (this if opt = max) or minimize (this if opt = min), the value $g(w, y)$, over all $y \in S(w)$.

DEFINITION 2.1. *Let* $\Phi = (S, g, \|\cdot\|, \|\cdot\|^*, \mathrm{opt})$ *be an optimization problem. The* optimum *value for instance* w *is denoted* $\Phi(w)$ *and defined by*

$$\Phi(w) \;=\; \begin{cases} \max_{y \in S(w)} g(w, y) & \text{if } \Phi \text{ is a maximization problem,} \\ \min_{y \in S(w)} g(w, y) & \text{if } \Phi \text{ is a minimization problem.} \end{cases}$$

The normalized optimum *is defined by* $\overline{\Phi}(w) = \Phi(w)/\|w\|^*$.

The above definition illustrates the role of the second norm, which is to normalize the optimum. Thus $\|\cdot\|^*$ will usually be chosen to make $0 \leq \overline{\Phi}(w) \leq 1$.

APPROXIMATION. An *approximation algorithm* for $\Phi = (S, g, \|\cdot\|, \|\cdot\|^*, \text{opt})$ is an algorithm A which on input w tries to output a number as close as possible to $\Phi(w)$. Unless otherwise indicated, an approximation algorithm runs in time polynomial in the length of w.

While the complexity of the algorithm is measured as a function of the length of the input, the approximation quality is often measured as a function of some other measure associated with the input. This is the first norm of w, denoted $\|w\|$. For example, for graph problems the first norm is typically the number of vertices in the graph.

The notion of an approximation algorithm achieving a certain approximation factor is different depending on whether it is a maximization problem or a minimization problem.

DEFINITION 2.2. *An approximation algorithm A for optimization problem $\Phi = (S, g, \|\cdot\|, \|\cdot\|^*, \text{opt})$ is said to achieve a factor $\mu(\cdot) \geq 1$ if for all instances w its output $A(w)$ satisfies*

- $\dfrac{\Phi(w)}{\mu(\|w\|)} \leq A(w) \leq \Phi(w)$ *if Φ is a maximization problem, or*
- $\Phi(w) \leq A(w) \leq \mu(\|w\|) \cdot \Phi(w)$ *if Φ is a minimization problem.*

Note that as per this definition, our convention is that an approximation factor is always a number at least 1. In some other sources, the approximation factor, at least in the case of minimization problems, is a number less than 1: they set it to the reciprocal of what we set it.

GAP PROBLEMS. We are interested in instances of an optimization problem for which the optimum is promised to be either "very high" or "very low." We capture this by associating to any optimization problem a promise problem, depending on a pair of "thresholds" c, s, both admissible functions of the first norm and satisfying $0 < s(\cdot) < c(\cdot)$. It is convenient to make the definition in terms of the normalized optimum rather than the optimum. In these cases, $c \leq 1$ typically holds. We consider maximization and minimization problems separately.

DEFINITION 2.3. *Let $\Phi = (S, g, \|\cdot\|, \|\cdot\|^*, \max)$ be a maximization problem, and let $0 < s(\cdot) < c(\cdot)$ be admissible functions of the first norm. Define*

$$\begin{aligned} Y &= \{\, w \,:\, \overline{\Phi}(w) \geq c(\|w\|) \,\}, \\ N &= \{\, w \,:\, \overline{\Phi}(w) < s(\|w\|) \,\}, \\ \textsf{Gap-}\Phi_{c,s} &= (Y, N)\,. \end{aligned}$$

The gap *of the promise problem is defined to be c/s.*

It is important that the inequality in the definitions of Y, N is strict in one case and not strict in the other. The same is true below, where we chose to maintain $s < c$ and so reversed the roles of s and c.

DEFINITION 2.4. *Let $\Phi = (S, g, \|\cdot\|, \|\cdot\|^*, \min)$ be a minimization problem, and let $0 < s(\cdot) < c(\cdot)$ be admissible functions of the first norm. Define*

$$\begin{aligned} Y &= \{\, w \,:\, \overline{\Phi}(w) \leq s(\|w\|) \,\}, \\ N &= \{\, w \,:\, \overline{\Phi}(w) > c(\|w\|) \,\}, \\ \textsf{Gap-}\Phi_{c,s} &= (Y, N)\,. \end{aligned}$$

The gap *of the promise problem is defined to be c/s.*

Each of the optimization problems we consider will give rise to a gap problem.

SHOWING NONAPPROXIMABILITY VIA GAPS. Hardness of approximation of some optimization problem is shown by reducing NP to $\mathsf{Gap}\text{-}\Phi_{c,s}$ via a (possibly randomized) Karp reduction. (This is called "producing hard gaps.") The following proposition says that we can show that Φ is hard to approximate within a factor g by showing $\mathrm{NP} \leq^{K} \mathsf{Gap}\text{-}\Phi_{c,s}$, for some s and $c = g \cdot s$, and the assumption under which the nonapproximability result holds depends on the type of reduction. It is this proposition that motivates the consideration of gap problems.

PROPOSITION 2.5. *Optimization problem Φ has no factor c/s approximation algorithm.*

Under the assumption:	*If it holds that:*
$\mathrm{P} \neq \mathrm{NP}$,	$\mathrm{NP} \leq_D^K \mathsf{Gap}\text{-}\Phi_{c,s}$,
$\mathrm{NP} \neq \mathrm{coRP}$,	$\mathrm{NP} \leq_R^K \mathsf{Gap}\text{-}\Phi_{c,s}$ *via a reduction with perfect completeness,*
$\mathrm{NP} \not\subseteq \mathrm{BPP}$,	$\mathrm{NP} \leq_R^K \mathsf{Gap}\text{-}\Phi_{c,s}$.

Proof. Let us illustrate by proving the first of the three claims, for the case that the problem is one of maximization. We proceed by contradiction. Given a (polynomial-time) algorithm A that achieves an approximation factor of $\mu = c/s$ for Φ, we present a polynomial-time algorithm B to decide L, where L is any language in NP. Let T be a (deterministic, Karp) reduction of L to $\mathsf{Gap}\text{-}\Phi_{c,s}$. On input x our algorithm B computes $w = T(x)$. Next it computes $\alpha = s(\|w\|) \cdot \|w\|^*$. Finally, B invokes A on w, outputs 1 (indicating $x \in L$) if $A(w) \geq \alpha$, and 0 otherwise (indicating $x \notin L$).

Since A runs in polynomial-time and the functions T, s, $\|\cdot\|$, $\|\cdot\|^*$ are polynomial-time computable (by hypothesis), the algorithm B runs in polynomial time. We claim that B is always right. To see this, let Y, N be the two parts of the promise problem $\mathsf{Gap}\text{-}\Phi_{c,s}$ as per Definition 2.3. Consider two cases.

First suppose $x \in L$. Then $w = T(x) \in Y$ because T is a correct reduction of L to $\mathsf{Gap}\text{-}\Phi_{c,s}$. So $\overline{\Phi}(w) \geq c(\|w\|)$ by Definition 2.3. However, starting from Definition 2.2 and simplifying, we have

$$\begin{aligned} A(w) \geq \frac{\Phi(w)}{c(\|w\|)/s(\|w\|)} &= \frac{\overline{\Phi}(w) \cdot \|w\|^*}{c(\|w\|)/s(\|w\|)} \\ \geq \frac{c(\|w\|) \cdot \|w\|^*}{c(\|w\|)/s(\|w\|)} &= s(\|w\|) \cdot \|w\|^* = \alpha\,. \end{aligned}$$

Thus B will output 1 as desired.

Now suppose $x \notin L$. Then $w = T(x) \in N$. So $\overline{\Phi}(w) < s(\|w\|)$ by Definition 2.3. Starting from Definition 2.2 and simplifying, we have

$$A(w) \leq \Phi(w) = \overline{\Phi}(w) \cdot \|w\|^* < s(\|w\|) \cdot \|w\|^* = \alpha\,.$$

Thus B will output 0 as desired.

The proofs for the other cases are similar (and thus omitted). □

2.4.2. Some optimization problems we consider. A *formula* is a set (actually a multiset) of clauses (i.e., or-clauses) over some set of literals. We consider various classes of formulae. In particular, 3-SAT formulae (at most three literals in each clause), E3-SAT formulae (exactly three different literals in each clause) and 2-SAT formulae (at most two literals in each clause). We use the generic notation X-SAT to stand for some specified class of formulae; thus the above correspond to $\mathrm{X} \in \{3, \mathrm{E3}, 2\}$. To each value of X we associate the following optimization problem.

Problem: MaxXSAT.
Instance: X-SAT formula φ.
Solutions: An assignment v which associates with any variable x of φ a Boolean value $v(x) \in \{0, 1\}$. (Not necessarily a satisfying assignment!)
Objective Function: The value of an assignment v is the number of clauses in φ that v makes true.
Norms: The norm $\|\varphi\|$ of formula φ is the number of clauses in it, and $\|\varphi\|^*$ is the same.
Type: Maximization.

In particular we have optimization problems Max2SAT, Max3SAT, MaxE3SAT, and their corresponding gap problems. A related optimization problem refers to sets of xor-clauses, or put differently, to a set of linear equations over GF(2).

Problem: MaxLinEq.
Instance: A sequence of linear equations over GF(2).
Solutions: An assignment v which associates with any variable x in the sequence a value $v(x) \in \mathrm{GF}(2)$.
Objective Function: The value of an assignment v is the number of equations that v satisfies.
Norms: Both norms are set to the number of equations in the instance.
Type: Maximization.

Problem: MaxCUT.
Instance: G, w, where $G = (V, E)$ is a graph and w: $E \to \mathcal{R}^+$ is a weight function.
Solutions: A *cut* $S, \overline{S}$ in G, meaning a partition $V = S \cup \overline{S}$ of V into disjoint sets.
Objective Function: The value of a cut is its weight $w(S, \overline{S})$, the sum of the weights of the edges with one endpoint in S and the other in $\overline{S}$.
Norms: $\|G, w\| = |V|$ and $\|G, w\|^* = \sum_{e \in E} w(e)$.
Type: Maximization.

We note that our inapproximability result (see section 4.3) holds also when w is an admissible integral function; that is, $w(e) \in \{1, \ldots, \text{ poly}(|E|)\}$, for every $e \in E$.

Problem: MinVC.
Instance: Graph $G = (V, E)$.
Solutions: A *vertex cover* in G, meaning a set $V' \subseteq V$ such that $V' \cap \{u, v\} \neq \emptyset$ for every $\{u, v\} \in E$.
Objective Function: The value of a vertex cover V' is its size, meaning the number of vertices in it.
Norms: $\|G\| = \|G\|^* = |V|$ is the number of vertices in the graph.
Type: Minimization.

Problem: MaxClique.
Instance: Graph $G = (V, E)$.
Solutions: A *clique* in G, meaning a set $C \subseteq V$ such that $\{u, v\} \in E$ for every pair u, v of distinct vertices in C.
Objective Function: The value of a clique C is its size, meaning the number of vertices in it.
Norms: $\|G\| = \|G\|^* = |V|$ is the number of vertices in the graph.
Type: Maximization.

Problem: ChromNum.
Instance: Graph $G = (V, E)$.
Solutions: A *coloring* of G, meaning a map $c\colon V \to \{1, \ldots, k\}$, for some k, such that $c(u) \neq c(v)$ for any $\{u, v\} \in E$.
Objective Function: The value of a coloring, c, is the number k of colors it uses.
Norms: $\|G\| = \|G\|^* = |V|$ is the number of vertices in the graph.
Type: Minimization.

As per our general notation, $\Phi(w)$ is the optimum for instance w of optimization problem Φ. Given the above, this means, for example, that MaxClique(G) is the maximum clique size in graph G; ChromNum(G) is the chromatic number of G; MinVC(G) is the minimum vertex cover size of G, etc. The corresponding normalized versions get bars overhead. We will use these notations in what follows.

By the above, MaxXSAT(φ) is the maximum number of simultaneously satisfiable clauses in φ. We abuse notation slightly by dropping the "X" and writing just MaxSAT(φ)—similarly for the normalized version.

2.4.3. History of approximability results for these problems.

Satisfiability problems. Max3SAT is the canonical MaxSNP complete problem [76]. A polynomial-time algorithm due to Yannakakis [85] approximates it to within a factor of $4/3 < 1.334$ (see [50] for an alternative algorithm). The best known polynomial-time algorithm for Max3SAT achieves a factor of $8/7$ (and is due to Karloff and Zwick [64], improving upon [51, 84]. For MaxE3SAT, which is also MaxSNP complete, a very simple algorithm achieves an approximation of $8/7 \leq 1.143$ (where $7/8$ is the expected fraction of clauses satisfied by a uniformly chosen assignment).

Max2SAT is also MaxSNP complete [49, 76]. This problem is particularly interesting because it has been the focus of recent improvements in the approximation factor attainable in polynomial time. Specifically, Goemans and Williamson [51] exhibited a polynomial-time algorithm achieving an approximation factor of $\frac{1}{0.878} \approx 1.139$, and subsequently Feige and Goemans [39] exhibited an algorithm achieving $\frac{1}{0.931} \approx 1.074$.

Nonapproximability results for MaxSNP problems begin with Arora et al. [8] who proved that there exists a constant $\epsilon > 0$ such that $\mathsf{Gap\text{-}3SAT}_{1,1-\epsilon}$ is NP-hard. They did this by providing a reduction from a given NP language L to the promise problem in question, constructed by encoding as a 3-SAT instance the computation of a $\mathrm{PCP}_{1,1/2}[\log, O(1)]$ verifier for an NP-complete language, the variables in the instance corresponding to bits in the proof string. The basic paradigm of their reduction has been maintained in later improvements.

Figure 6 depicts the progress. Improvements (in the constant value of the nonapproximability factor) begin with Bellare et al. [21]. They used Hadamard code based inner verifiers following [8]. They also introduced a framework for better analysis and improved some previous analyses; we exploit in particular their better analyses of linearity testing (cf. section 3.5) and of Freivalds's matrix multiplication test (cf. Lemma 3.16). The improvement of Feige and Kilian [41] was obtained via new proof systems; that of [23] by use of the canonicity property of constant-prover proofs and some optimizations. (See section 2.2.3 for a discussion of the role of constant-prover proofs in this context.)

Garey, Johnson, and Stockmeyer [49] had provided, as early as 1976, a reduction of Max3SAT to Max2SAT which showed that if the former is nonapproximable within $(k + 1)/k$, then the latter is nonapproximable within $(7k + 1)/(7k)$. With the best

Due to	Assuming	Factor	Technique
[8]	$P \neq NP$	some constant	$NP \subseteq PCP_{1,1/2}[\log, O(1)]$; reduction of this to Max3SAT.
[21]	$\widetilde{P} \neq \widetilde{NP}$	94/93	Framework; better analyses; uses proof systems of [67, 43].
[21]	$P \neq NP$	113/112	New four-prover proof systems.
[41]	$P \neq NP$	94/93	New two-prover proof systems.
[23]	$\widetilde{P} \neq \widetilde{NP}$	66/65	Canonicity and some optimizations.
[23]	$P \neq NP$	73/72	Canonicity and some optimizations.
This paper	$P \neq NP$	27/26	Long code and new proof systems.

FIG. 6. *Nonapproximability results for* Max3SAT *indicating the factor shown hard and the assumption under which they were achieved.*

previous nonapproximability factor for Max3SAT (namely 66/65) we would only get a 456/455 factor nonapproximability for Max2SAT. In fact, even using our new Max3SAT result we would get a hardness factor of only 185/184. See section 4.2 for our results.

LINEAR EQUATIONS. The MaxLinEq problem is known to be MaxSNP complete (see [29] or [77]).

We remark that the problem of *maximizing* the number of satisfiable equations should not be confused with the "complementary" problem of *minimizing* the number of violated constraints, investigated by Arora et al. [7]. Also, the case of maximum satisfiable linear constraints over larger fields (of size q) has been considered by Amaldi and Kann [5], who show that this problem is hard to approximate to within a factor of q^ϵ for some universal $\epsilon > 0$. See section 4.2.2 for our results.

MAX CUT. In 1976, Sahni and Gonzales [80] gave a simple 2-approximation algorithm for this problem. Recently, in a breakthrough result, Goemans and Williamson [51] gave a new algorithm which achieves a ratio of $\frac{1}{0.878} = 1.139$ for this problem. On the other hand, [76] give an approximation preserving reduction from Max3SAT to MaxCUT. Combined with [8] this shows that there exists a constant $\alpha > 1$ such that approximating MaxCUT within a factor of α is NP-hard. No explicit bounds were given since and even using the best known hardness results for MAX3SAT, one suspects that the bound for MaxCUT would not be very large, since the reduction uses constructions of constant degree expanders, etc. See section 4.3 for our results.

VERTEX COVER. There is a simple polynomial-time algorithm to approximate MinVC in unweighted graphs within a factor of 2. The algorithm, due to F. Gavril (cf. [48]) consists of taking all vertices which appear in a maximal matching of the graph. For weighted graphs, Bar-Yehuda and Even [13] and Hochbaum [58] gave algorithms achieving the same approximation factor. The best known algorithm today achieves a factor only slightly better, namely, $2 - (\log\log|V|)/(2\log|V|)$ [14, 74].

Evidence to the hardness of approximating MinVC was given by Bar-Yehuda and Moran who showed that, for every $k \geq 2$ and $\epsilon > 0$, a $1 + \frac{1}{k} - \epsilon$ approximator for (finding) a minimum vertex cover would yield an algorithm for coloring $(k+1)$-colorable graphs using only logarithmically many colors [15]. The version of MinVC in which one restricts attention to graphs of bounded degree, is MaxSNP complete [76]. In particular they provide a reduction from Max3SAT. Combined with [8] this implies the existence of a constant $\delta > 0$ such that approximating MinVC within a factor of $1 + \delta$ is hard unless P = NP. No explicit value of δ has been stated

until now. Indeed, the value that could be derived, even using the best existing nonapproximability results for Max3SAT, will be very small because of the cost of the reduction of [76], which first reduces Max3SAT to its bounded version using expanders, and then reduces this to the bounded-degree version of MinVC. See section 5.2 for our results.

MaxClique. The best known polynomial-time approximation algorithm for MaxClique achieves a factor of only $N^{1-o(1)}$ [28], scarcely better than the trivial factor of N. There is not even a heuristic algorithm that is conjectured to do better. (The Lovász theta function had been conjectured to approximate the MaxClique size within $\sqrt{N}$, but this conjecture was disproved by Feige [37].)

Prior to 1991, no nonapproximability results on MaxClique were known. In 1991 the connection to proofs was made by Feige et al. [40]. The FGLSS reduction says that $\mathrm{PCP}_{1,e}[\,\mathsf{coins} = r\,;\,\mathsf{query} = q\,]$ Karp reduces to $\mathsf{Gap}\text{-MaxClique}_{c,s}$ via a reduction running in time $\mathrm{poly}(2^{r+q})$, and with the gap c/s being a function of (r, q, and) the error e. In applying the reduction one works with PCP classes containing NP. One obtains a result stating that MaxClique has no polynomial-time approximation algorithm achieving a certain factor, under an assumption about the deterministic time complexity of NP (the time complexity depends on r, q, and the factor on these, but, most importantly, on the error e). In particular, these authors were able to "scale down" the proof system of [11] to indicate strong nonapproximability factors of $2^{\log^{1-\epsilon} N}$ for all $\epsilon > 0$, assuming NP is not in quasi-polynomial deterministic time. They also initiated work on improving the factors and assumptions via better proof systems. The best result in their paper is indicated in Figure 7.

Arora and Safra [9] reduced the randomness complexity of a PCP verifier for NP to logarithmic; they showed $\mathrm{NP} = \mathrm{PCP}_{1,1/2}[\,\mathsf{coins} = \log\,;\,\mathsf{query} = \sqrt{\log N}\,]$. On the other hand, it is easy to see that random bits can be recycled for error reduction via the standard techniques of [2, 30, 59]. The consequence was the first NP-hardness result for MaxClique approximation. The corresponding factor was $2^{\sqrt{\log N}}$.

Arora et al. [8] showed that $\mathrm{NP} = \mathrm{PCP}_{1,1/2}[\,\mathsf{coins} = \log\,;\,\mathsf{query} = O(1)\,]$, which implied that there exists an $\epsilon > 0$ for which approximating MaxClique within N^ϵ was NP-complete. The number of queries was unspecified but was estimated to be $\approx 10^4$, and so $\epsilon \approx 10^{-4}$. Later work has focused on increasing the constant value of ϵ in the exponent.

In later work a slightly tighter form of the FGLSS reduction due to [25, 86] has been used. It says that $\mathrm{PCP}_{1,1/2}[\,\mathsf{coins} = r\,;\,\mathsf{query}_{\mathrm{av}} = q_{\mathrm{av}}\,]$ reduces, via a randomized Karp reduction, to $\mathsf{Gap}\text{-MaxClique}_{c,s}$ for some c, s satisfying $c(N)/s(N) = N^{1/(1+q_{\mathrm{av}})}$, and with the running time of the reduction being $\mathrm{poly}(2^r)$. (We assume $q_{\mathrm{av}} = O(1)$ for simplicity, and omit factors of $N^{\epsilon'}$ where $\epsilon' > 0$ can be arbitrarily small, here and in the following.) Thus the hardness factor was tied to the (average) number of queries required to get soundness error $1/2$. Meanwhile the assumption involved the probabilistic rather than deterministic time complexity of NP—it would be $\mathrm{NP} \not\subseteq \widetilde{\mathrm{coRP}}$ if $r = \mathrm{polylog}(n)$ and $\mathrm{NP} \neq \mathrm{coRP}$ if $r = \log(n)$.

The improved proof systems of [21] obtain significantly smaller query complexity; they showed $\mathrm{NP} \subseteq \mathrm{PCP}_{1,1/2}[\,\mathsf{coins} = \mathrm{polylog}; \mathsf{query} = 24\,]$ and $\mathrm{NP} \subseteq \mathrm{PCP}_{1,1/2}[\,\mathsf{coins} = \log\,;\,\mathsf{query} = 29\,]$. This leads to their hardness results shown in Figure 7. However, significantly reducing the (average) number of bits queried seemed hard.

As observed by Feige and Kilian, the performance of the FGLSS reduction actually depends on the free-bit complexity, which may be significantly smaller than the query

Due to	Factor	Assumption
[40]	$2^{\log^{1-\epsilon} N}$ for any $\epsilon > 0$	NP $\not\subseteq \widetilde{\text{P}}$
[9]	$2^{\sqrt{\log N}}$	P $\neq$ NP
[8]	N^{ϵ} for some $\epsilon > 0$	P $\neq$ NP
[21]	$N^{1/30}$	NP $\neq$ coRP
[21]	$N^{1/25}$	NP $\not\subseteq$ co$\widetilde{\text{RP}}$
[41]	$N^{1/15}$	NP $\neq$ coRP
[23]	$N^{1/6}$	P $\neq$ NP
[23]	$N^{1/4}$	NP $\not\subseteq$ co$\widetilde{\text{RP}}$
This paper	$N^{1/4}$	P $\neq$ NP
This paper	$N^{\frac{1}{3}}$	NP $\neq$ coRP

FIG. 7. *Progress in the project of increasing the nonapproximability factors for MaxClique.*

complexity [41]. Namely, the factor in the above mentioned reduction is $N^{1/(1+f)}$ where f is the free-bit complexity. They observed that the proof system of [21] has free-bit complexity 14, yielding a $N^{1/15}$ hardness of approximation factor.

The notion of amortized free bits was introduced by Bellare and Sudan [23]. They observed that the performance of the reduction depended, in fact, on this quantity and that the factor was $N^{1/(1+\bar{f})}$, where $\bar{f}$ is the amortized free-bit complexity. They then showed that NP $\subseteq \overline{\text{FPCP}}[\text{polylog}, 3]$. This lead to a $N^{1/4}$ hardness factor assuming NP $\neq$ co$\widetilde{\text{RP}}$. See section 7 for our results.

CHROMATIC NUMBER. The first hardness result for the chromatic number is due to Garey and Johnson [47]. They showed that if P $\neq$ NP, then there is no polynomial-time algorithm that can achieve a factor less than 2. This remained the best result until the connection to proof systems, and the above mentioned results, emerged.

Hardness results for the chromatic number were obtained via reduction from MaxClique. An N^{ϵ} factor hardness for MaxClique translates into a N^{δ} factor hardness for the Chromatic number[3], with $\delta = \delta(\epsilon)$ a function of ϵ. In all reductions $\delta(\epsilon) = \min\{h(\epsilon), h(0.5)\}$, for some function h. The bigger h, the better the reduction.

The first reduction, namely that of Lund and Yannakakis [71], obtained $h(\epsilon) = \epsilon/(5-4\epsilon)$. Via the MaxClique hardness results of [9, 8], this implies that the chromatic number is hard to approximate within N^{δ} for some $\delta > 0$. But, again, δ is very small. Improvements to δ were derived by both improvements to ϵ and improvements to the function h used by the reduction.

A subsequent reduction of Khanna, Linial, and Safra [66] is simpler, but in fact slightly less efficient, having $h(\epsilon) = \epsilon/(5+\epsilon)$. A more efficient reduction is given by [23]—they present a reduction obtaining $h(\epsilon) = \epsilon/(3-2\epsilon)$. Our $N^{\frac{1}{3}}$ hardness for Clique would yield, via this, a $N^{1/7}$ hardness for the chromatic number. But more recently an even more efficient reduction has become available, namely that of Fürer [45]. This reduction achieves $h(\epsilon) = \epsilon/(2-\epsilon)$, and thereby we get our $N^{\frac{1}{5}}$ hardness.

Following the appearance of our results, Feige and Kilian [42] have introduced a new approach to showing hardness of approximability of ChromNum. See discussion in section 1.5.

[3]Actually, all the reductions presented here make assumptions regarding the structure of the graph and hence do not directly yield the hardness results stated here. However, as a consequence of some results from this paper, we are able to remove the assumptions made by the earlier papers and hence present those results in a simpler form. See section 8.4 for details.

Randomized and derandomized error reduction. As mentioned above, randomized and derandomized error reduction techniques play an important role in obtaining the best clique hardness results via the FGLSS method. Typically, one first reduces the error so that its logarithm relates to the query (or free-bit) complexity and so that the initial randomness cost can be ignored (as long as it was logarithmic). (Otherwise, one would have needed to construct proof systems which also minimize this parameter, i.e., the constant factor in the logarithmic randomness complexity.)

The randomized error reduction method originates in the work of Berman and Schnitger [25], where it is applied to the Clique-Gap promise problem. An alternative description is given by Zuckerman [86]. Another alternative description, carried out in the proof system, is presented in section 11.

The derandomized error-reduction method consists of applying general, derandomized, error-reduction techniques to the proof system setting.[4] The best method, known as the "expander walk" technique, is due to Ajtai, Komlos, and Szemeredi [2] (see also [30, 59]). It is easy to see that it is applicable to the pcp context. (The usage of these methods in the pcp context begins with [9].) It turns out that the (constant) parameters of the expander, specifically the ratio $\rho \stackrel{\text{def}}{=} \frac{\log_2 d}{\log_2 \lambda}$, where d is the degree of the expander and λ is the second eigenvalue (of its adjacency matrix), play an important role here. In particular, $\rho - 1$ determines how much we lose with respect to the randomized error reduction (e.g., $\text{NP} \in \overline{\text{FPCP}}[\log, f]$ translates to a hardness factor of $N^{\frac{1}{1+f}}$ under $\text{NP} \not\subseteq \text{BPP}$ and to a hardness factor of $N^{\frac{1}{\rho+f}}$ under $\text{NP} \neq \text{P}$). Thus the Ramanujan expander of Lubotzky, Phillips, and Sarnak [70] plays an important role yielding $\rho \approx 2$ (cf. Proposition 11.4), which is the best possible (when using the expander walk technique).

Part I: New proof systems and nonapproximability results.

3. The long code and its machinery.

3.1. New PCPs and hardness results: Overview and guidemap. The starting point for all of our proof systems is a two-prover proof system achieving arbitrarily small but fixed constant error with logarithmic randomness and constant answer size, as provided by Raz [79]. This proof system has the property that the answer of the second prover is supposed to be a predetermined function of the answer of the first prover. Thus, verification in it amounts to checking that the first answer satisfies some predicate and that the second answer equals the value obtained from the first answer. Following the "proof composition" paradigm of Arora and Safra [9], the proof string provided to the PCP verifier will consist of "encodings" of the answers of the two provers under a suitable code. The PCP verifier will then check these encodings. As usual, we will check both that these encodings are valid and that they correspond to answers which would have been accepted by the original verifier.

Our main technical contribution is a new code, called the *long code*, and the means to check it. The long code of an n-bit information word a is the sequence of 2^{2^n} bits consisting of the values of all possible Boolean functions at a. The long code is certainly a disaster in terms of coding efficiency, but it has big advantages in the context of proof verification because it carries enormous amounts of data about a.

[4] An alternative approach, applicable to the Gap-Clique problem and presented in [3], is to "derandomize" the graph product construction of [25].

The difficulty will be to check that a prover claiming to write the long code of some string a is really doing so.

The long code is described in section 3.3. In section 3.5 we provide what we call the "atomic" tests for this code. These tests and their analyses are instrumental to all that follows. Section 3.4 is also instrumental to all that follows. This section sets up the framework for recursive proof checking which is used in all the later proof systems.

The atomic tests are exploited in section 4.1 to construct a verifier that queries the proof at three locations and performs one of two simple checks on the answers obtained. These simple checks are implemented by gadgets of the MaxSNP problem at hand, yielding the nonapproximability results. Section 4.2 presents gadgets which are CNF formulae of the corresponding type and section 4.3 presents MaxCUT gadgets. The nonapproximability results for Max3SAT, MaxE3SAT, Max2SAT, and MaxCUT follow. The verifier of section 4.1 benefits from another novel idea which is referred to as *folding* (see section 3.3). Folding contributes to the improved results for Max3SAT, MaxE3SAT, Max2SAT, and MaxCUT, but not to the results regarding MaxClique (and Chromatic Number).

A reasonable nonapproximability result for MinVC (Minimum Vertex Cover) can be obtained by the above methodology, but a better result is obtained by constructing a different verifier, for NP languages, that uses exactly two free bits. This verifier is then used to create a reduction of NP to MinVC via the FGLSS and standard Karp reductions. This approach is presented in section 5, where we try to minimize the soundness error attainable using exactly two free bits.

In section 6 we minimize the number of bits queried in a PCP to attain soundness error 1/2—the result is not of direct applicability, but it is intriguing to know how low this number can go.

We then turn to MaxClique (and Chromatic Number). In section 7.1 we provide the "iterated" tests. (Here the atomic tests are invoked (sequentially) many times, but these invocations are not independent of each other.) This leads to a proof system in which the number of *amortized free bits* used is two. We then draw the implications for MaxClique (and Chromatic Number). A reader interested only in the (amortized) free-bit and MaxClique results can proceed directly from section 3.5 to section 7.

The improvement in the complexities of the proof systems is the main source of our improved nonapproximability results. In addition we also use (for the MaxSAT, MaxCUT, and MinVC problems) a recent improvement in the analysis of linearity testing [17], and introduce (problem specific) gadgets which represent the various tests of the resulting PCP system.

3.2. Preliminaries to the long code. Here $\Sigma = \{0, 1\}$ will be identified with the finite field of two elements, the field operations being addition and multiplication modulo two. If X and Y are sets, then $\mathsf{Map}(X, Y)$ denotes the set of all maps of X to Y. For any integer m we regard Σ^m as a vector space over Σ so that strings and vectors are identified. If $a \in \Sigma^m$, then $a^{(i)}$ denotes its ith bit. Similarly, if f is any function with range Σ^m, then $f^{(i)}$ denotes the ith bit of its output.

LINEARITY. Let G, H be groups. A map $f\colon G \to H$ is *linear* if $f(x + y) = f(x) + f(y)$ for all $x, y \in G$. Let $\textsc{Lin}(G, H)$ denote the set of all linear maps of G to H. When $G = \Sigma^n$ and $H = \Sigma$, a function $f\colon G \to H$ is linear if and only if there exists $a \in \Sigma^n$ such that $f(x) = \sum_{i=1}^{n} a^{(i)} x^{(i)}$ for all $x \in \Sigma^n$.

DISTANCE. The *distance* between functions f_1, f_2 defined over a common finite domain D is

$$\mathrm{Dist}(f_1, f_2) \;=\; \mathrm{Pr}_{x \stackrel{R}{\leftarrow} D}\left[f_1(x) \neq f_2(x)\right].$$

Functions f_1, f_2 are *ϵ-close* if $\mathrm{Dist}(f_1, f_2) < \epsilon$. If f maps a group G to a group H we denote by $\mathrm{Dist}(f, \textsc{Lin})$ the minimum, over all $g \in \textsc{Lin}(G, H)$, of $\mathrm{Dist}(f, g)$. (Note that the notation does not specify G, H, which will be evident from the context.) We are mostly concerned with the case where G is a vector space V over Σ and H is Σ. Notice that in this case we have $\mathrm{Dist}(f, \textsc{Lin}) \leq 1/2$ for all $f\colon V \to \Sigma$.

Boolean functions. Let l be an integer. We let $\mathcal{F}_l \stackrel{\text{def}}{=} \mathsf{Map}(\Sigma^l, \Sigma)$ be the set of all maps of Σ^l to Σ. We regard $\mathcal{F}_l$ as a vector space (of dimension 2^l) over Σ. Addition and multiplication of functions are defined pointwise.

We let $\mathcal{L}_m \subseteq \mathcal{F}_m$ be the set $\textsc{Lin}(\Sigma^m, \Sigma)$ of linear functions of Σ^m to Σ and let $\mathcal{L}_m^* = \mathcal{L}_m - \{0\}$ be the nonzero linear functions.

Let $g \in \mathcal{F}_m$ and $\vec{f} = (f_1, \ldots, f_m) \in \mathcal{F}_l^m$. Then $g \circ \vec{f}$ denotes the function in $\mathcal{F}_l$ that assigns the value $g(f_1(x), \ldots, f_m(x))$ to $x \in \Sigma^l$.

The monomial basis. For each $S \subseteq [l]$ we let $\chi_S \in \mathcal{F}_l$ be the monomial corresponding to S, defined for $x \in \Sigma^l$ by

$$\chi_S(x) \;=\; \textstyle\prod_{i \in S} x^{(i)} \;.$$

The empty monomial, namely $\chi_\emptyset$, is defined to be the constant-one function (i.e., $\chi_\emptyset(x) \;=\; \bar{1}$, for all $x \in \Sigma^l$). The functions $\{\chi_S\}_{S \subseteq [l]}$ form a basis for the vector space $\mathcal{F}_l$ which we call the *monomial basis.* This means that, for each $f \in \mathcal{F}_l$, there exists a unique vector $\mathcal{C}(f) = (\, C_f(S)\,)_{S \subseteq [l]} \in \Sigma^{2^l}$ such that

$$f \;=\; \textstyle\sum_{S \subseteq [l]} C_f(S) \cdot \chi_S \;. \tag{5}$$

The expression on the right-hand side above is called the *monomial series* for f, and the members of $\mathcal{C}(f)$ are called the *coefficients of f* with respect to the monomial basis. We note that $\mathcal{C}\colon \mathcal{F}_l \to \Sigma^{2^l}$ is a bijection. (The monomial basis is reminiscent of the Fourier basis, but the two are actually different.)

3.3. Evaluation operators, the long code, and folding.

Evaluation operators. Let $a \in \Sigma^l$. We define the map $E_a\colon \mathcal{F}_l \to \Sigma$ by $E_a(f) = f(a)$ for all $f \in \mathcal{F}_l$. We say that a map $A\colon \mathcal{F}_l \to \Sigma$ is an *evaluation operator* if there exists some $a \in \Sigma^l$ such that $A = E_a$. We now provide a useful characterization of evaluation operators. First we need a definition.

Definition 3.1 (respecting the monomial basis). *A map $A\colon \mathcal{F}_l \to \Sigma$ is said to* respect the monomial basis *if*

(1) $A(\chi_\emptyset) = 1$ *and*

(2) $\forall\, S, T \subseteq [l] \;\;:\;\; A(\chi_S) \cdot A(\chi_T) = A(\chi_{S \cup T})$.

Proposition 3.2 (characterization of the evaluation operator). *A map $\tilde{A}\colon \mathcal{F}_l \to \Sigma$ is an evaluation operator if and only if it is linear and respects the monomial basis.*

Proof. Let $a \in \Sigma^l$. It is easy to see that E_a is linear: $E_a(f + g) = (f + g)(a) = f(a) + g(a) = E_a(f) + E_a(g)$. It is also easy to see that E_a respects the monomial basis. Firstly, we have $E_a(\chi_\emptyset) = \chi_\emptyset(a) = 1$. Next, for every $S, T \subseteq [l]$,

$$E_a(\chi_S) \cdot E_a(\chi_T) \;=\; \chi_S(a) \cdot \chi_T(a) \;=\; \textstyle\prod_{i \in S} a^{(i)} \;\cdot\; \prod_{i \in T} a^{(i)} \;.$$

However, $x^2 = x$ for any $x \in \Sigma$, so

$$\textstyle\prod_{i \in S} a^{(i)} \;\cdot\; \prod_{i \in T} a^{(i)} \;=\; \prod_{i \in S \cup T} a^{(i)} \;=\; \chi_{S \cup T}(a) \;=\; E_a(\chi_{S \cup T}).$$

Now we turn to the converse. Let $\tilde{A}\colon \mathcal{F}_l \to \Sigma$ be linear and respect the monomial basis. For $i = 1, \ldots, l$, let $a_i \stackrel{\text{def}}{=} \tilde{A}(\chi_{\{i\}})$, and let $a \stackrel{\text{def}}{=} a_1, \ldots, a_l$. We claim that $\tilde{A} = E_a$. The proof is as follows. We first claim that

$$\forall\, S \subseteq [l] \quad : \quad \tilde{A}(\chi_S) = \chi_S(a)\,. \tag{6}$$

Since $\tilde{A}$ respects the monomial basis we have $\tilde{A}(\chi_\emptyset) = 1$ which in turn equals $\chi_\emptyset(a)$, proving Eq. (6) for $S = \emptyset$. To establish Eq. (6) for $S = \{i_1, \ldots, i_t\} \neq \emptyset$, we write

$$\tilde{A}(\chi_S) \;=\; \tilde{A}\left(\chi_{\{i_1\}\cup\cdots\cup\{i_t\}}\right) \;=\; \textstyle\prod_{j=1}^{t} \tilde{A}(\chi_{\{i_j\}}) \;=\; \prod_{j=1}^{t} a_{i_j} \;=\; \chi_S(a)\,,$$

where the second equality is due to the fact that $\tilde{A}$ respects the monomial basis. This establishes Eq. (6). Now for any $f \in \mathcal{F}_l$ we can use the linearity of $\tilde{A}$ to see that

$$\begin{aligned}\tilde{A}(f) = \tilde{A}\left(\textstyle\sum_S C_f(S)\cdot\chi_S\right) \;&=\; \textstyle\sum_S C_f(S)\cdot\tilde{A}(\chi_S)\\ = \textstyle\sum_S C_f(S)\cdot\chi_S(a) \;&=\; f(a) \;=\; E_a(f)\,.\end{aligned}$$

Thus $\tilde{A} = E_a$. □

THE LONG CODE. Intuitively, the encoding of $a \in \{0,1\}^l$ (via the long code) is the 2^{2^l} bit string which in position $f \in \mathcal{F}_l$ stores the bit $f(a)$. The long code is thus an extremely "redundant" code, encoding an l-bit string by the values, at a, of *all* functions in $\mathcal{F}_l$.

DEFINITION 3.3 (long code). *The* long code $E\colon \Sigma^l \to \mathsf{Map}(\mathcal{F}_l, \Sigma)$ *is defined for any* $a \in \Sigma^l$ *by* $E(a) = E_a$.

In some natural sense E is the longest possible code: E is the longest code which is not repetitive (i.e., does not have two positions which are identical in all codewords).

We let $\mathrm{Dist}(A, \textsc{Eval}) = \min_{a\in\Sigma^l} \mathrm{Dist}(A, E_a)$ be the distance from A to a closest codeword of E. It is convenient to define $E^{-1}(A) \in \Sigma^l$ as the lexicographically least $a \in \Sigma^l$ such that $\mathrm{Dist}(A, E_a) = \mathrm{Dist}(A, \textsc{Eval})$. Notice that if $\mathrm{Dist}(A, \textsc{Eval}) < 1/4$, then there is exactly one $a \in \Sigma^l$ such that $\mathrm{Dist}(A, E_a) = \mathrm{Dist}(A, \textsc{Eval})$, and so $E^{-1}(A)$ is this a.

The long code is certainly a disaster in terms of coding efficiency, but it has a big advantage in the context of proof verification. Consider, for example, the so-called "circuit test" (i.e., testing that the answer of the first prover satisfies some predetermined predicate/circuit). In this context one needs to check that the codeword encodes a string which satisfies a predetermined predicate (i.e., the codeword encodes some $w \in \{0,1\}^n$, which satisfies $h(w) = 0$, for some predetermined predicate h). The point is that the value of this predicate appears explicitly in the codeword itself, and furthermore it can be easily "self-corrected" by probing the codeword for the values of the functions f and $f + h$, for a uniformly selected function $f : \{0,1\}^n \to \{0,1\}$ (as all these values appear explicitly in the codeword). Actually, the process of verifying, via self-correction, that the value under h is zero can be incorporated into the task of checking the validity of the codeword; this is done by the notion of "$(h, 0)$-folding" (see below). The fact that we can avoid testing whether the codeword encodes a string which satisfies a given function (or that this testing does not cost us anything) is the key to the complexity improvements in our proof systems (over previous proof systems in which a "circuit test" was taking place and had a considerable cost).

FOLDING. The intuition behind the notion we will now define is like this. When A is the long code of some string x for which it is known that $h(x) = b$, for some function h and bit b, then half of the bits of A become redundant because they can

be computed from the other half via $A(f) = A(f+h) - b$. This phenomenon enables us to reduce the proof checking complexity. To capture it we now define the notion of folding.

Fix $\prec$ to be some canonical, polynomial-time computable total order (reflexive, antisymmetric, transitive) on the set $\mathcal{F}_l$. Given functions $A\colon \mathcal{F}_l \to \Sigma$ and $h \in \mathcal{F}_l \setminus \{\bar{0}\}$ (i.e., h is not the constant function $\bar{0}$) and bit $b \in \Sigma$, the (h,b)-*folding* of A is the function $A_{(h,b)}\colon \mathcal{F}_l \to \Sigma$ given by

$$A_{(h,b)}(f) = \begin{cases} A(f) & \text{if } f \prec h+f \\ A(f+h) - b & \text{otherwise.} \end{cases}$$

(Notice that the above is well defined for any $h \neq \bar{0}$.) For sake of technical simplicity (see Definition 3.9), we define the $(\bar{0},0)$-folding of A to be A itself; namely, $A_{(\bar{0},0)}(f) = A(f)$, for every $f \in \mathcal{F}_l$. As shown below, the (h,b)-folding of a function A is forced to satisfy $A_{(h,b)}(f+h) = A_{(h,b)}(f) + b$, for every $f \in \mathcal{F}_l$ (whereas A itself may not necessarily satisfy these equalities). Before proving this, let us generalize the notion of folding to folding over several, specifically two, functions $h_1, h_2 \in \mathcal{F}_l$ (and bits $b_1, b_2 \in \Sigma$).

DEFINITION 3.4 (folding). *Let $f, h_1, h_2 \in \mathcal{F}_l$. The* (h_1,h_2)-span *of f, denoted* $\textsc{span}_{h_1,h_2}(f)$*, is defined as the set* $\{f + \sigma_1 h_1 + \sigma_2 h_2 : \sigma_1, \sigma_2 \in \Sigma\}$*. Let* $\textsc{mincoef}_{h_1,h_2}(f)$ *be the pair (σ_1,σ_2) of elements of Σ for which $f+\sigma_1 h_1+\sigma_2 h_2$ is the smallest function (according to $\prec$) in* $\textsc{span}_{h_1,h_2}(f)$*. Let $A\colon \mathcal{F}_l \to \Sigma$. Assume that h_1, h_2 are distinct and nonzero, and let $b_1, b_2 \in \Sigma$. The* folding *of A* over (h_1,b_1) and (h_2,b_2)*, denoted $A_{(h_1,b_1),(h_2,b_2)}$, is defined for every $f \in \mathcal{F}_l$ by*

$$A_{(h_1,b_1),(h_2,b_2)}(f) = A(f + \sigma_1 h_1 + \sigma_2 h_2) - \sigma_1 b_1 - \sigma_2 b_2 \ ,$$

where $(\sigma_1,\sigma_2) = \textsc{mincoef}_{h_1,h_2}(f)$.

The definition extends naturally to the following two cases. In case $(h_1,b_1) = (h_2,b_2)$, folding over the two (identical) pairs is defined as folding over one pair. In case $h_1 \equiv \bar{0}$ and $b_1 = 0$, folding over both (h_1,b_1) and (h_2,b_2) is defined as folding over (h_2,b_2). Note that folding over two pairs is invariant under the order between the pairs; namely, $A_{(h_1,b_1),(h_2,b_2)} \equiv A_{(h_2,b_2),(h_1,b_1)}$. Finally, observe that a function $A\colon \mathcal{F}_l \to \Sigma$ that is folded over two functions (i.e., over both (h_1,b_1) and (h_2,b_2)) is folded over each of them (i.e., over each (h_i,b_i)).

PROPOSITION 3.5 (folding forces equalities). *Let $A\colon \mathcal{F}_l \to \Sigma$, $h_1, h_2 \in \mathcal{F}_l$, and $b_1, b_2 \in \Sigma$ (with $b_i = 0$ in case $h_i \equiv \bar{0}$). Then, for every $f \in \mathcal{F}_l$,*

$$A_{(h_1,b_1),(h_2,b_2)}(f+h_1) = A_{(h_1,b_1),(h_2,b_2)}(f) + b_1.$$

Proof. By definition, $A_{(h_1,b_1),(h_2,b_2)}(f) = A(f + \sigma_1 h_1 + \sigma_2 h_2) - \sigma_1 b_1 - \sigma_2 b_2$, where the function $f + \sigma_1 h_1 + \sigma_2 h_2$ is the smallest function in $\textsc{span}_{h_1,h_2}(f)$. Since $\textsc{span}_{h_1,h_2}(f+h_1) \equiv \textsc{span}_{h_1,h_2}(f)$, we have $A_{(h_1,b_1),(h_2,b_2)}(f+h_1) = A(f + \sigma_1 h_1 + \sigma_2 h_2) - (\sigma_1 - 1)b_1 - \sigma_2 b_2$. The claim follows. □

As a corollary to the above (combined with the self-correcting paradigm [27]), we get the following proposition.

PROPOSITION 3.6 (folding and the evaluation operator). *Let $A\colon \mathcal{F}_l \to \Sigma$, $h \in \mathcal{F}_l$, $b \in \Sigma$, and $a \in \Sigma^l$. Suppose that for any $f \in \mathcal{F}_l$ it is the case that $A(f+h) = A(f)+b$. Then* $\mathrm{Dist}(A, E_a) < 1/2$ *implies $h(a) = b$. Consequently, if* $\mathrm{Dist}(A_{(h,b),(h',b')}, E_a) < 1/2$*, then $h(a) = b$, provided $b = 0$ if $h \equiv \bar{0}$.*

Proof. By the hypothesis, we have $A(h+f) = A(f)+b$, for every $f \in \mathcal{F}_l$. Suppose that $\mathrm{Dist}(A, E_a) < 1/2$. Then, noting that E_a is linear and applying a self-correction process (cf. Corollary 3.14 below), we get $E_a(h) = b$. Using the definition of the evaluator operator (i.e., $E_a(h) = h(a)$), we have $h(a) = b$. The consequence for $A_{(h,b),(h',b')}$ follows since by Proposition 3.5 we have $A_{(h,b),(h',b')}(f + h) = A_{(h,b),(h',b')}(f) + b$ for any $f \in \mathcal{F}_l$. □

The verifiers constructed below make virtual access to "folded" functions rather than to the functions themselves. Virtual access to a folding of A is implemented by actual accessing A itself according to the definition of folding (e.g., say one wants to access $A_{(h,0)}$ at f; then one determines whether or not $f \prec h + f$ and accesses either $A(f)$ or $A(f + h)$, accordingly). One benefit of folding in our context is illustrated by Proposition 3.6; in case a (h, b)-folded function is close to a codeword (in the long code), we infer that the codeword encodes a string a satisfying $h(a) = b$. We will see that folding (the long code) over $(h, 0)$ allows us to get rid of a standard ingredient in proof verification—the so-called "circuit test."

In the following, we will use folding over the pairs $(h, 0)$ and $(\bar{1}, 1)$, where $h \in \mathcal{F}_l$ is an arbitrary function (typically not identically zero) and $\bar{1}$ is the constant-one function. Folding over $(\bar{1}, 1)$ allows us to simplify the "codeword" test (w.r.t. the long code).

3.4. Recursive verification of proofs. This section specifies the basic structure of proof construction and in particular provides the definitions of the notions of inner and outer verifiers which will be used throughout. It is useful to understand these things before proceeding to the tests.

Overview. The constructions of efficient proofs that follow will exploit the notion of recursive verifier construction due to Arora and Safra [9]. We will use just one level of recursion. We first define a notion of a *canonical outer verifier* whose intent is to capture two-prover, one-round proof systems [24] having certain special properties; these verifiers will be our starting point. We then define a canonical inner verifier. Recursion is captured by an appropriate definition of a composed verifier whose attributes we relate to those of the original verifiers in Theorem 3.12.

The specific outer verifier we will use is one obtained by a recent work of Raz [79]. We will construct various inner verifiers based on the long code and the tests in sections 3.5 and 7.1. Theorem 3.12 will be used ubiquitously to combine the two.

Comparison with previous work. For a history and a better understanding of the role of constant-prover proof systems in this context, see section 2.4.3. In comparison, our definition of outer verifiers (below) asks for almost the same canonicity properties as in [23]. (The only difference is that they have required σ to be a projection function, whereas we can deal with an arbitrary function. But we don't take advantage of this fact.) In addition we need answer sizes of $\log \log n$ as opposed to the $O(\log n)$ of previous methods, for reasons explained below. This means that even the (modified) [67, 43] type proofs won't suffice for us. We could use the three-prover modification of [41] but the cost would wipe out our gain. Luckily this discussion is moot since we can use the recent result of Raz [79] to provide us with a canonical two-prover proof having logarithmic randomness, constant answer size, and any constant error. This makes an ideal starting point. To simplify the definitions below we insisted on constant answer size and two provers from the start.

The inner verifiers used in all previous works are based on the use of the Hadamard code constructions of [8]. (The improvements mentioned above are obtained by checking this same code in more efficient ways.) We instead use a new code, namely, the

long code, as the basis of our inner verifiers. Note that the codewords (in the long code) have length double exponential in the message, explaining our need for $\log \log n$ answer sizes in the outer verifier. We also incorporate into the definitions the new idea of folding which we will see means that we don't need a circuit test (a hint toward this fact is already present in the definition of a good inner verifier).

3.4.1. Outer verifiers. As mentioned above, outer verifiers will model special kinds of two-prover, one-round proof systems. We think of the verifier as being provided with a pair of (nonboolean) proof oracles π, π_1, and allowed one query to each. The desired properties concern the complexity of the system and a certain behavior in the checking of the proof, as we now describe.

Let $r_1, s, s_1 \colon \mathcal{Z}^+ \to \mathcal{Z}^+$ and let l and l_1 be positive integers. An (l, l_1)-*canonical outer verifier* V_{outer} takes as input $x \in \Sigma^n$ and has oracle access to a pair of proofs $\bar{\pi} \colon [s(n)] \to \Sigma^l$ and $\bar{\pi}_1 \colon [s_1(n)] \to \Sigma^{l_1}$. It does the following.

(1) Picks a random string R_1 of length $r_1(n)$.
(2) Computes, as a function of x and R_1, queries $q \in [s(n)]$ and $q_1 \in [s_1(n)]$, and a (circuit computing a) function $\sigma \colon \Sigma^l \to \Sigma^{l_1}$ (which is determined by x and R_1); determines, based on x and q, a function $h \colon \Sigma^l \to \Sigma$ (and computes an appropriate representation of it).
(We stress that h does not depend on R_1 but only on q and x).
(3) Lets $a = \bar{\pi}(q)$ and $a_1 = \bar{\pi}_1(q_1)$.
(4) If $h(a) \neq 0$ then rejects.
(5) If $\sigma(a) \neq a_1$ then rejects.
(6) Otherwise accepts.

We call s, s_1 the *proof sizes* for V_{outer} and r_1 the *randomness* of V_{outer}. Both s and s_1 are bounded above by 2^{r_1}.

Recall that by the conventions in section 2, $\mathtt{ACC}[V_{\text{outer}}^{\bar{\pi},\bar{\pi}_1}(x)]$ denotes the probability, over the choice of R_1, that V_{outer} accepts, and $\mathtt{ACC}[V_{\text{outer}}(x)]$ denotes the maximum of $\mathtt{ACC}[V_{\text{outer}}^{\bar{\pi},\bar{\pi}_1}(x)]$ over all possible proofs $\bar{\pi}, \bar{\pi}_1$.

DEFINITION 3.7 (goodness of outer verifier). *Outer verifier* V_{outer} *is* ϵ-good *for language* L *if for all* x *it is the case that*
(1) $x \in L$ *implies* $\mathtt{ACC}[V_{\text{outer}}(x)] = 1$.
(2) $x \notin L$ *implies* $\mathtt{ACC}[V_{\text{outer}}(x)] \leq \epsilon$.

Employing the FRS method [44] to any PCP(log,O(1)) system for NP (e.g., [8]) one gets a canonical verifier which is δ-good for some $\delta < 1$. (Roughly, the method is to take the given pcp system, send all queries to one oracle, and, as a check, send a random query to the other oracle.) Using the parallel repetition theorem of Raz [79], we obtain our starting point.

LEMMA 3.8 (construction of outer verifiers). *Let* $L \in$ NP. *Then for every* $\epsilon > 0$ *there exist positive integers* l, l_1 *and* c *such that there exists an* (l, l_1)-*canonical outer verifier which is* ϵ-*good for* L *and uses randomness* $r(n) = c \log_2 n$.

Actually, Raz's theorem [79] enables one to assert that l, l_1 and c are $O(\log \epsilon^{-1})$; but we will not need this fact. Also, the function σ determined by this verifier is always a projection, but we don't use this fact either.

3.4.2. Inner verifiers. Inner verifiers are designed to efficiently verify that the encoding of answers, which a (canonical) outer verifier expects to see, indeed satisfies the checks which this outer verifier performs. Typically, the inner verifier performs a combination of a codeword test (i.e., tests that each oracle is indeed a proper encoding relative to a fixed code—in our case the long code), a projection test (i.e., that the decoding of the second answer corresponds to the value of σ applied to the decoding

of the first), and a "circuit test" (i.e., that the decoding of the first answer evaluates to 0 under the function h).

Let $r_2, l, l_1 \in \mathcal{Z}^+$. An (l, l_1)-*canonical inner verifier* V_{inner} takes as inputs functions $\sigma\colon \Sigma^l \to \Sigma^{l_1}$ and $h \in \mathcal{F}_l$. (It may also take additional inputs, depending on the context.) It has oracle access to a pair of functions $A\colon \mathcal{F}_l \to \Sigma$ and $A_1\colon \mathcal{F}_{l_1} \to \Sigma$ and uses r_2 random bits. The parameters $\delta_1, \delta_2 > 0$ in the following should be thought of as extremely small: in our constructions, they are essentially 0 (see comment below).

DEFINITION 3.9 (goodness of inner verifier). *An inner verifier* V_{inner} *is* $(\rho, \delta_1, \delta_2)$-good *if, for all* σ, h *as above, the following hold.*

(1) *Suppose* $a \in \Sigma^l$ *is such that* $h(a) = 0$. *Let* $a_1 = \sigma(a) \in \Sigma^{l_1}$. *Then*

$$\mathtt{ACC}\left[\, V_{\text{inner}}^{E_a, E_{a_1}}(\sigma, h)\,\right] = 1\ .$$

(2) *Suppose* A, A_1 *are such that* $\mathtt{ACC}\left[\, V_{\text{inner}}^{A, A_1}(\sigma, h)\,\right] \geq \rho$. *Then there exists* $a \in \Sigma^l$ *such that*

(2.1) $\text{Dist}(A_{(h,0),(\bar{1},1)}, E_a) < 1/2 - \delta_1$.

(2.2) $\text{Dist}(A_1, E_{\sigma(a)}) < 1/2 - \delta_2$.

We stress that, although the inner verifier has access to the oracle A (and the hypothesis in condition (2) of Definition 3.9 refers to its computations with oracle A), the conclusion in condition (2.1) refers to A folded over both $(h, 0)$ and $(\bar{1}, 1)$, where $\bar{1}$ is the constant-one function. (Typically, but not necessarily, the verifier satisfying Definition 3.9 accesses the virtual oracle $A_{(h,0),(\bar{1},1)}$ by actual access to A according to the definition of folding.) Furthermore, by Proposition 3.6, condition (2.1) implies that $h(a) = 0$. (Thus, there is no need to explicitly require that $h(a) = 0$ in order to make Theorem 3.12 work.) We comment that the upper bounds in conditions (2.1) and (2.2) are chosen to be the largest ones which still allow us to prove Theorem 3.12 (below). Clearly, the complexity of the inner verifier decreases as these bounds increase. This is the reason for setting δ_1 and δ_2 to be extremely small. We stress that this optimization is important for the MaxSNP results but not for the MaxClique result. In the latter case, we can use δ_i's greater than $\frac{1}{4}$ which simplifies a little the analysis of the composition of verifiers (below).

Remark 3.10 (a tedious one). The above definition allows h to be identically zero (although this case never occurs in our constructions nor in any other reasonable application). This is the reason that we had to define folding over (0,0) as well. An alternative approach would have been to require $h \not\equiv 0$ and assert that this is the case with respect to the outer verifier of Lemma 3.8.

3.4.3. Composition of verifiers. We now describe the *canonical* composition of a (canonical) outer verifier with a corresponding (canonical) inner verifier. Let V_{outer} be an (l, l_1)-canonical outer verifier with randomness r_1 and proof sizes s, s_1. Let V_{inner} be an (l, l_1)-canonical inner verifier with randomness r_2. Their composed verifier $\langle V_{\text{outer}}, V_{\text{inner}}\rangle$ takes as input $x \in \Sigma^n$ and has oracle access to proofs $\pi\colon [s(n)] \times \mathcal{F}_l \to \Sigma$ and $\pi_1\colon [s_1(n)] \times \mathcal{F}_{l_1} \to \Sigma$. It behaves as follows:

—Picks random strings for both V_{outer} and V_{inner}; namely, picks a random string R_1 of length $r_1(n)$ and a random string R_2 of length $r_2(n)$.

—Computes queries q and q_1 and functions σ and h as V_{outer} would compute them given x, R_1.

—Outputs $V_{\text{inner}}^{A, A_1}(\sigma, h; R_2)$, where $A(\cdot) = \pi(q, \cdot)$ and $A_1(\cdot) = \pi_1(q_1, \cdot)$.

The randomness complexity of the composed verifier is $r_1 + r_2$ whereas its query and free-bit complexities equal those of V_{inner}.

We show how the composed verifier $\langle V_{\text{outer}}, V_{\text{inner}}\rangle$ inherits the goodness of the V_{outer} and V_{inner}. To do so we need the following lemma. It is the counterpart of a claim in [21, Lemma 3.5] and will be used in the same way. The lemma is derived from a coding theory bound which is a slight extension of the bounds in [73, Chap. 7] (see Appendix).

LEMMA 3.11. *Suppose* $0 < \delta \leq 1/2$ *and* $A\colon \mathcal{F}_l \to \Sigma$. *Then there are at most* $1/(4\delta^2)$ *codewords that have distance less than* $1/2 - \delta$ *from* A. *That is,*

$$\left|\{\, a \in \Sigma^l \,:\, \text{Dist}(A, E_a) \leq 1/2 - \delta \,\}\right| \;\leq\; \frac{1}{4\delta^2}\,.$$

Furthermore, for $\delta > 1/4$ *the above set contains at most one string.*

Proof. We know that E_a is linear for any a (cf. Proposition 3.2). So it suffices to upper bound the size of the set

$$\mathcal{A} \;=\; \{\, X \in \mathit{Lin}(\mathcal{F}_l, \Sigma) \,:\, \text{Dist}(A, X) \leq 1/2 - \delta \,\}\,.$$

This set has the same size as

$$\mathcal{B} \;=\; \{\, X - A \,:\, X \in \mathit{Lin}(\mathcal{F}_l, \Sigma) \text{ and } \text{Dist}(A, X) \leq 1/2 - \delta \,\}\,.$$

Let $n = 2^{2^l}$ and identify $\mathsf{Map}(\mathcal{F}_l, \Sigma)$ with Σ^n in the natural way. Let $w(\cdot)$ denote the Hamming weight. Now note that $Z = X - A \in \mathcal{B}$ implies $w(Z)/n = \text{Dist}(X, A) \leq 1/2 - \delta$. Furthermore, if $Z_1 = X_1 - A$ and $Z_2 = X_2 - A$ are in $\mathcal{B}$, then $\text{Dist}(Z_1, Z_2) = \text{Dist}(X_1, X_2)$ and the latter is $1/2$ if $X_1 \neq X_2$, since X_1, X_2 are linear. Thus, $\mathcal{B}$ is a set of binary vectors of length n, each of weight at most $(0.5 - \delta)n$, and any two of distance at least $0.5n$ apart. Invoking Lemma A.1 (with $\alpha = \delta$ and $\beta = 0$), we upper bound the size of $\mathcal{B}$ as desired. Finally, when $\delta > 1/4$, the triangle inequality implies that we cannot have $a_1 \neq a_2$ so that $\text{Dist}(A, E_{a_i}) \leq 1/2 - \delta < 1/4$ for both $i = 1, 2$. □

In some applications of the following theorem, $\delta_1, \delta_2 > 0$ will first be chosen to be so small that they may effectively be thought of as 0. (This is done in order to lower the complexities of the inner verifiers.) Once the δ_i's are fixed, ϵ will be chosen to be so much smaller (than the δ_i's) that $\epsilon/(16\delta_1^2\delta_2^2)$ may be thought of as effectively 0. The latter explains why we are interested in outer verifiers which achieve a constant, but arbitrarily small, error ϵ. For completeness we provide a proof following the ideas of [9, 8, 21].

THEOREM 3.12 (the composition theorem). *Let* V_{outer} *be an* (l, l_1)*-canonical outer verifier. Suppose it is* ϵ*-good for* L. *Let* V_{inner} *be an* (l, l_1)*-canonical inner verifier that is* $(\rho, \delta_1, \delta_2)$*-good. Let* $V = \langle V_{\text{outer}}, V_{\text{inner}}\rangle$ *be the composed verifier, and let* $x \in \Sigma^*$. *Then*

(1) *if* $x \in L$, *then* $\mathtt{ACC}[\,V(x)\,] = 1$.

(2) *if* $x \notin L$, *then* $\mathtt{ACC}[\,V(x)\,] \leq \rho + \epsilon/16\delta_1^2\delta_2^2$.

For $\delta_1, \delta_2 > 1/4$ *the upper bound in* (2) *can be improved to* $\rho + \epsilon$.

The latter case (i.e., $\delta_1, \delta_2 > 1/4$) suffices for the MaxClique results.

Proof. Let $n = |x|$, and let s, s_1 denote the proof sizes of V_{outer}.

Suppose $x \in L$. By Definition 3.7 there exist proofs $\bar{\pi}\colon [s(n)] \to \Sigma^l$ and $\bar{\pi}_1\colon [s_1(n)] \to \Sigma^{l_1}$ such that $\mathtt{ACC}[\,V_{\text{outer}}^{\bar{\pi},\bar{\pi}_1}(x)\,] = 1$. Let $\pi\colon [s(n)] \times \mathcal{F}_l \to \Sigma$ be defined by $\pi(q, f) = E_{\bar{\pi}(q)}(f)$. (In other words, replace the l bit string $\bar{\pi}(q)$ with its 2^{2^l} bit encoding under the long code, and let the new proof provide access to the bits in this encoding). Similarly let $\pi_1\colon [s_1(n)] \times \mathcal{F}_{l_1} \to \Sigma$ be defined by $\pi_1(q_1, f_1) = E_{\bar{\pi}_1(q_1)}(f_1)$.

The reader may easily verify that the item (1) properties in Definitions 3.7 and 3.9 (of the outer and inner verifiers, respectively) imply that $\mathtt{ACC}[\,V^{\pi,\pi_1}(x)\,] = 1$.

Now suppose $x \notin L$. Let $\pi\colon [s(n)] \times \mathcal{F}_l \to \Sigma$ and let $\pi_1\colon [s_1(n)] \times \mathcal{F}_{l_1} \to \Sigma$ be proof strings for V. We will show that $\mathtt{ACC}[\,V^{\pi,\pi_1}(x)\,] \le \rho + \epsilon/(16\delta_1^2\delta_2^2)$. Since π, π_1 are arbitrary, this will complete the proof.

We set $N_1 = \lfloor 1/(4\delta_1^2) \rfloor$ and $N_2 = \lfloor 1/(4\delta_2^2) \rfloor$ (with $N_1 = 1$ if $\delta_1 > 1/4$ and $N_2 = 1$ if $\delta_2 > 1/4$). The idea to show $\mathtt{ACC}[\,V^{\pi,\pi_1}(x)\,] \le \rho + N_1 N_2 \cdot \epsilon$ is as follows. We will first define a collection of N_1 proofs $\bar\pi^1, \ldots, \bar\pi^{N_1}$ and a collection of N_2 proofs $\bar\pi_1^1, \ldots, \bar\pi_1^{N_2}$ so that each pair $(\bar\pi^i, \bar\pi_1^j)$ is a pair of oracles for the *outer* verifier. Next we will partition the random strings R_1 of the *outer* verifier into two categories, depending on the performance of the *inner* verifier on the inputs (i.e., the functions σ, h and the oracles A, A_1) induced by R_1. On the "bad" random strings of the *outer* verifier, the *inner* verifier will accept with probability at most ρ; on the "good" ones, we will use the soundness of the *inner* verifier to infer that the *outer* verifier accepts under some oracle pair $(\bar\pi^i, \bar\pi_1^j)$, for $i \in [N_1]$ and $j \in [N_2]$. The soundness of the *outer* verifier will be used to bound the probability of such acceptances.

We now turn to the actual analysis. We define N_1 proofs $\bar\pi^1, \ldots, \bar\pi^{N_1}\colon [s(n)] \to \Sigma^l$ as follows. Fix $q \in [s(n)]$ and let $A = \pi(q, \cdot)$. Let $B_q = \{a \in \Sigma^l : \mathrm{Dist}(A_{(h,0),(\bar 1,1)}, E_a) \le 1/2 - \delta_1\}$. (Notice that for this set to be well defined we use the fact that h is well defined given q.) Note that $|B_q| \le N_1$ by Lemma 3.11. Order the elements of B_q in some canonical way, adding dummy elements to bring the number to exactly N_1, so that they can be written as $a^1(q), \ldots, a^{N_1}(q)$. Now set $\bar\pi^i(q) = a^i(q)$ for $i = 1, \ldots, N_1$. In a similar fashion we define $\bar\pi_1^j(q_1) = a_1^j(q_1)$ for $j = 1, \ldots, N_2$, where each $a_1^j = a_1^j(q_1)$ satisfies $\mathrm{Dist}(\pi_1(q_1, \cdot), E_{a_1^j}) \le 1/2 - \delta_2$.

Let R_1 be a random string of V_{outer}. We say that R_1 is *good* if

$$\mathtt{ACC}[\,V_{\mathrm{inner}}^{\pi(q,\cdot),\pi_1(q_1,\cdot)}(\sigma, h)\,] \;\ge\; \rho\,,$$

where q, q_1, σ, h are the queries and functions specified by R_1. If R_1 is not good we say it is *bad*. The claim that follows says that if R_1 is good then there is some choice of the above defined proofs which leads the outer verifier to accept on coins R_1.

CLAIM. *Suppose R_1 is good. Then there is an $i \in [N_1]$ and a $j \in [N_2]$ such that $V_{\mathrm{outer}}^{\bar\pi^i, \bar\pi_1^j}(x; R_1) = 0$.*

Proof. Let q, q_1, σ, h be the queries and functions specified by R_1. Let $A = \pi(q, \cdot)$ and $A_1 = \pi_1(q_1, \cdot)$ (be the oracles accessed by the inner verifier). Since R_1 is good we have $\mathtt{ACC}[\,V_{\mathrm{inner}}^{A,A_1}(\sigma, h)\,] \ge \rho$. So by item 2 of Definition 3.9 there exists $a \in \Sigma^l$ such that $\mathrm{Dist}(A_{(h,0),(\bar 1,1)}, E_a) < 1/2 - \delta_1$ and $\mathrm{Dist}(A_1, E_{\sigma(a)}) < 1/2 - \delta_2$. Let $a_1 = \sigma(a)$. Since $\mathrm{Dist}(A_{(h,0),(\bar 1,1)}, E_a) \le 1/2 - \delta_1$ it must be the case that $a \in B_q$, and hence there exists $i \in [N_1]$ such that $a = \bar\pi^i(q)$. Similarly $\mathrm{Dist}(A_1, E_{\sigma(a)}) < 1/2 - \delta$ implies that there is some $j \in [N]$ such that $a_1 = \bar\pi_1^j(q_1)$. By Proposition 3.6 we have $h(a) = 0$, and we have $\sigma(a) = a_1$ by (the above) definition. Now, by definition of the (execution of the) canonical outer verifier, $V_{\mathrm{outer}}^{\bar\pi^i, \bar\pi_1^j}(x; R_1) = 0$ holds. □

By conditioning we have $\mathtt{ACC}[\,V^{\pi,\pi_1}(x)\,] \le \alpha + \beta$, where

$$\alpha = \Pr_{R_1}[R_1 \text{ is good}],$$
$$\beta = \Pr_{R_1,R_2}[\,V^{\pi,\pi_1}(x; R_1 R_2) = 0 \mid R_1 \text{ is bad}\,]\,.$$

The definition of badness of R_1 implies $\beta \le \rho$. On the other hand, we can use the

claim to see that

$$\begin{aligned}\alpha &\le \Pr_{R_1}\left[\exists i \in [N_1]\ ,\ j \in [N_2]\ :\ V_{\text{outer}}^{\bar{\pi}^i,\bar{\pi}_1^j}(x;R_1)=0\right]\\ &\le \textstyle\sum_{i=1}^{N_1}\sum_{j=1}^{N_2}\Pr_{R_1}\left[V_{\text{outer}}^{\bar{\pi}^i,\bar{\pi}_1^j}(x;R_1)=0\right]\\ &\le N_1N_2\cdot\epsilon\ ,\end{aligned}$$

where the last inequality holds by the soundness of V_{outer} (i.e., item 2 of Definition 3.7). Using the bound on N_1 and N_2, the proof is concluded. □

3.5. The atomic tests.

MOTIVATION. Our constructions of proofs systems will use the outer verifier of Lemma 3.8, composed via Theorem 3.12 with inner verifiers to be constructed. The brunt of our constructions is the construction of appropriate inner verifiers. The inner verifier will have oracle access to a function $A\colon \mathcal{F}_l \to \Sigma$ and a function $A_1\colon \mathcal{F}_{l_1} \to \Sigma$. In all of our applications, A is supposed to be a folding of an encoding of the answer a of the first prover (in a two-prover proof system) and A_1 is supposed to be the encoding of the answer a_1 of the second prover. The verifier will perform various tests to determine whether these claims are true. The design of these tests is the subject of this subsection.

The atomic tests we provide here will be used directly in the proof systems for showing nonapproximability of Max3SAT, Max2SAT, and MaxCUT. Furthermore, they are also the basis of iterated tests which will lead to proof systems of amortized free-bit complexity ≈ 2, which in turn are used for the MaxClique and Chromatic Number results. We remark that for the applications to the above-mentioned MaxSNP problems (but not for the application to MaxClique) it is important to have the best possible analysis of our atomic tests, and what follows strives to this end. We stress that the exposition and analysis of these tests, in this subsection, are independent of the usage of the codes in our proof systems.

TESTING FOR A CODEWORD. The first task that concerns us is to design a test which, with high probability, passes if and only if A is close to an evaluation operator (i.e., a valid codeword). The idea is to exploit the characterization of Proposition 3.2. Thus we will perform (on A) a linearity test and then a "respect of monomial basis" (RMB) test. Linearity testing is well understood, and we will use the test of [27] with the analyses of [27, 21, 17]. The main novelty is the respect of monomial basis test.

CIRCUIT AND PROJECTION. Having established that A is close to some evaluation operator E_a, we now want to test two things. The first is that $h(a) = 0$ for some predetermined function h. This test which would normally be implemented by "self-correction" (i.e., evaluating $h(a)$ by uniformly selecting $f \in \mathcal{F}_l$ and computing $A(f + h) - A(f)$) is not needed here since in our applications we will use an $(h, 0)$-folding of A instead of A. Thus, it is left to test that the two oracles are consistent in the sense that A_1 is not too far from an evaluation operator which corresponds to $\sigma(a)$ for some predetermined function σ.

SELF-CORRECTION. The following self-correction lemma is due to [27] and will be used throughout.

LEMMA 3.13 (self-correction lemma [27]). *Let $A, \tilde{A}\colon \mathcal{F}_l \to \Sigma$ with $\tilde{A}$ linear, and let $x = \text{Dist}(A, \tilde{A})$. Then for every $g \in \mathcal{F}_l$,*

$$\Pr_{f \overset{R}{\leftarrow} \mathcal{F}_l}\left[A(f+g) - A(f) = \tilde{A}(g)\right] \ \ge\ 1 - 2x\ .$$

The atomic tests. Here $A\colon \mathcal{F}_l \to \Sigma$ and $A_1\colon \mathcal{F}_{l_1} \to \Sigma$ are the objects being tested. The tests also take additional inputs or parameters: below $f, f_1, f_2, f_3 \in \mathcal{F}_l$; $g \in \mathcal{F}_{l_1}^m$; and $\sigma\colon \Sigma^l \to \Sigma^{l_1}$.

LinTest$(A; f_1, f_2)$ (Linearity Test).
If $A(f_1) + A(f_2) = A(f_1 + f_2)$, then output 0 else output 1.

MBTest$(A; f_1, f_2, f_3)$ (Respecting-Monomial-Basis Test).
If $A(f_1) = 0$, then check if $A(f_1 \cdot f_2 + f_3) = A(f_3)$.
Otherwise (i.e., $A(f_1) = 1$) then check if $A(f_1 \cdot f_2 + f_2 + f_3) = A(f_3)$.
Output 0 if the relevant check succeeded, else output 1.

ProjTest$_\sigma(A, A_1; f, g)$ (Projection Test).
If $A_1(g) = A(g \circ \sigma + f) - A(f)$, then output 0, else output 1.

The passing probabilities. These are the probabilities we are interested in:

$$\begin{aligned}
\textsc{LinPass}(A) &= \Pr_{f_1, f_2 \overset{R}{\leftarrow} \mathcal{F}_l}[\,\mathbf{LinTest}(A; f_1, f_2) = 0\,],\\
\textsc{MBPass}(A) &= \Pr_{f_1, f_2, f_3 \overset{R}{\leftarrow} \mathcal{F}_l}[\,\mathbf{MBTest}(A; f_1, f_2, f_3) = 0\,],\\
\textsc{ProjPass}_\sigma(A, A_1) &= \Pr_{f \overset{R}{\leftarrow} \mathcal{F}_l\,;\, g \overset{R}{\leftarrow} \mathcal{F}_{l_1}}[\,\mathbf{ProjTest}_\sigma(A, A_1; f, g) = 0\,].
\end{aligned}$$

FIG. 8. *The atomic tests and their passing probabilities.*

Proof.

$$\begin{aligned}
&\Pr_{f \overset{R}{\leftarrow} \mathcal{F}_l}\left[A(f+g) - A(f) = \tilde{A}(g)\right]\\
&\geq \Pr_{f \overset{R}{\leftarrow} \mathcal{F}_l}\left[A(f+g) = \tilde{A}(f+g) \ \text{ and } \ A(f) = \tilde{A}(f)\right]\\
&\geq 1 - \left(\Pr_{f \overset{R}{\leftarrow} \mathcal{F}_l}\left[A(f+g) \neq \tilde{A}(f+g)\right] + \Pr_{f \overset{R}{\leftarrow} \mathcal{F}_l}\left[A(f) \neq \tilde{A}(f)\right]\right).
\end{aligned}$$

However, each of the probabilities in the last expression equals x. □

COROLLARY 3.14. *Let $A, \tilde{A}\colon \mathcal{F}_l \to \Sigma$ with $\tilde{A}$ linear, and suppose $x \overset{\text{def}}{=} \text{Dist}(A, \tilde{A}) < 1/2$. Suppose also that $A(f + h) = A(f) + b$, for some fixed $h \in \mathcal{F}_l$, $b \in \Sigma$, and every $f \in \mathcal{F}_l$. Then $\tilde{A}(h) = b$.*

Proof. By the hypothesis, we have $A(f + h) - A(f) = b$ for all functions f. Thus, we can write

$$\Pr_{f \overset{R}{\leftarrow} \mathcal{F}_l}\left[A(f+h) - A(f) = \tilde{A}(h)\right] \;=\; \Pr_{f \overset{R}{\leftarrow} \mathcal{F}_l}\left[b = \tilde{A}(h)\right],$$

but the right-hand side (and hence the left) is either 0 or 1 (as both h and b are fixed). However, by Lemma 3.13 the left-hand side is bounded below by $1 - 2x > 0$ and so the corollary follows. □

CONVENTION. All our tests output a bit with 0 standing for accept and 1 for reject.

3.5.1. Atomic linearity test. The *atomic linearity test* shown in Figure 8 is the one of Blum, Luby, and Rubinfeld [27]. We want to lower bound the probability $1 - \textsc{LinPass}(A)$ that the test rejects, when its inputs f_1, f_2 are chosen at random, as

a function of $x = \mathrm{Dist}(A, \textsc{Lin})$. The following lemma, due to Bellare et. al. [17], gives the best known lower bound.

LEMMA 3.15 (see [17]). *Let $A\colon \mathcal{F}_l \to \Sigma$ and let $x = \mathrm{Dist}(A, \textsc{Lin})$. Then $1 - \textsc{LinPass}(A) \geq \Gamma_{\mathrm{lin}}(x)$, where the function $\Gamma_{\mathrm{lin}}\colon [0, 1/2] \to [0, 1]$ is defined as follows:*

$$\Gamma_{\mathrm{lin}}(x) \stackrel{\mathrm{def}}{=} \begin{cases} 3x - 6x^2 & 0 \leq x \leq 5/16, \\ 45/128 & 5/16 \leq x \leq 45/128, \\ x & 45/128 \leq x \leq 1/2. \end{cases}$$

The above lower bound is composed of three different bounds with "phase transitions" at $x = \frac{5}{16}$ and $x = \frac{45}{128}$. It was shown in [17] (see below) that this combined lower bound is close to the best one possible.

PERSPECTIVE. The general problem of linearity testing as introduced and studied by Blum, Luby, and Rubinfeld [27] is stated as follows. Given a function $A\colon G \to H$, where G, H are groups, obtain a lower bound on r_A as a function of x_A, where

$$r_A = \Pr_{a,b \stackrel{R}{\leftarrow} G}[A(a) + A(b) \neq A(a + b)]$$
$$x_A = \mathrm{Dist}(A, \textsc{Lin}) .$$

Blum, Luby, and Rubinfeld showed that $r_A \geq \frac{2}{9} x_A$, for every A. Their analysis was used in the proof system and Max3SAT nonapproximability result of [8]. Interest in the tightness of the analysis began with Bellare et al. [21], with the motivation of improving the Max3SAT nonapproximability results. They showed that $r_A \geq 3x_A - 6x_A^2$, for every A. This establishes the first segment of the lower bound quoted above (i.e., of the function Γ_{lin}). Also, it is possible to use [27] to show that $r_A \geq \frac{2}{9}$ when $x_A \geq \frac{1}{4}$. Putting these together implies a two-segment lower bound with phase transition at the largest root of the equation $3x - 6x^2 = \frac{2}{9}$ (i.e., at $\frac{1}{4} + \frac{\sqrt{33}}{36}$). This lower bound was used in the Max3SAT analyses of [21] and [23].

However, for our applications (i.e., linearity testing over $\mathcal{F}_l$ as in Lemma 3.15), the case of interest is when the underlying groups are $G = \mathrm{GF}(2)^n$ and $H = \mathrm{GF}(2)$ (since $\mathcal{F}_l$ may be identified with $\mathrm{GF}(2)^n$ for $n = 2^l$). The work of Bellare et al. [17] focused on this case and improved the bound on r_A for the case $x_A \geq \frac{1}{4}$, where $A\colon \mathrm{GF}(2)^n \to \mathrm{GF}(2)$. Specifically, they showed that $r_A \geq 45/128$ for $x_A \geq \frac{1}{4}$ which establishes the second segment of Γ_{lin}. They also showed that $r_A \geq x_A$, for every $A\colon \mathrm{GF}(2)^n \to \mathrm{GF}(2)$. Combining the three lower bounds, they have derived the three-segment lower bound stated in Lemma 3.15.

The optimality of the above analysis has been demonstrated as well in [17]. Essentially,[5] for every $x \leq \frac{5}{16}$ there are functions $A\colon \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ witnessing $r_A = \Gamma_{\mathrm{lin}}(x_A)$ with $x_A = x$. For the interval $(\frac{5}{16}, 1/2]$, no tight results are known. Instead, Bellare et al. [17] reports of computer constructed examples of functions $A\colon \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ with x_A in every interval $[\frac{k}{100}, \frac{k+1}{100}]$, for $k = 32, 33, \ldots, 49$, and $r_A < \Gamma_{\mathrm{lin}}(x_A) + \frac{1}{20}$. Furthermore, they showed that there exist such functions with both x_A and r_A arbitrarily close to $1/2$.

3.5.2. Monomial basis test. Having determined that A is close to linear, the *atomic respect of monomial basis* test makes sure that the linear function close to A respects the monomial basis. Let us denote the latter function (i.e., the linear function closest to A) by $\tilde{A}$. Recalling Definition 3.1 we need to establish two things, namely, that $\tilde{A}(\chi_\emptyset) = 1$ and that $\tilde{A}(\chi_S) \cdot \tilde{A}(\chi_T) = \tilde{A}(\chi_{S \cup T})$, for every $S, T \subseteq [l]$. Recall that we do not have access to $\tilde{A}$ but rather to A; still, the self-correction lemma provides

[5] Actually, the statement holds only for x's which are integral multiples of 2^{-n}.

an obvious avenue to bypass the difficulty provided $\mathrm{Dist}(A, \tilde{A}) < \frac{1}{4}$. This would have yielded a solution, but quite a wasteful one (though sufficient for the MaxClique and Chromatic Number results). Instead, we adopt the following more efficient procedure.

Firstly, by considering only oracles folded over $(\bar{1}, 1)$, we need not check that $\tilde{A}(\chi_\emptyset) = 1$. (This follows by combining Corollary 3.14 and the fact that the $(\bar{1}, 1)$-folded oracle A satisfies $A(f + \bar{1}) = A(f) + 1$, for all $f \in \mathcal{F}_l$.) Secondly, we test that $\tilde{A}(\chi_S) \cdot \tilde{A}(\chi_T) = \tilde{A}(\chi_{S \cup T})$, for every $S, T \subseteq [l]$, by taking random linear combinations of the S's and T's to be tested. Such linear combinations are nothing but uniformly selected functions in $\mathcal{F}_l$. Namely, we wish to test $\tilde{A}(f) \cdot \tilde{A}(g) = \tilde{A}(f \cdot g)$, where f and g are uniformly selected in $\mathcal{F}_l$. Since A is close to $\tilde{A}$, we can inspect $A(f)$ (resp., $A(g)$) rather than $\tilde{A}(f)$ (resp., $\tilde{A}(g)$) with little harm. However, $f \cdot g$ is not uniformly distributed (when f and g are uniformly selected in $\mathcal{F}_l$) and thus self-correction will be applied here. The resulting test is

$$A(f_1) \cdot A(f_2) = A(f_1 \cdot f_2 + f_3) - A(f_3). \tag{7}$$

This test was analyzed in a previous version of this work [20]; specifically, this test was shown to reject a folded oracle A, with $\tilde{A}$ (the linear function closest to A) which does not respect the monomial basis, with probability at least $(1-2x) \cdot (\frac{3}{8} - x + \frac{x^2}{2}) = \frac{3}{8} - \frac{7}{4}x + \frac{5}{2}x^2 - x^3$, where $x = \mathrm{Dist}(A, \tilde{A})$. Here we present an adaptive version of the above test, which performs even better. We observe that if $A(f_1) = 0$, then there is no need to fetch $A(f_2)$ (since the left-hand side of Eq. (7) is zero regardless of $A(f_2)$). Thus, we merely test whether $A(f_1 \cdot f_2 + f_3) - A(f_3) = 0$. But what should be done if $A(f_1) = 1$? In this case we may replace f_1 by $f_1 + \bar{1}$ (yielding $A(f_1 + \bar{1}) = A(f_1) + 1 = 0$) and test whether $A((f_1 + \bar{1}) \cdot f_2 + f_3) - A(f_3) = 0$. The resulting test is depicted in Figure 8.

A TECHNICAL LEMMA. First we recall the following lemma of [21] which provides an improved analysis of Freivalds' matrix multiplication test in the special case when the matrices are symmetric with common diagonal.

LEMMA 3.16 (symmetric matrix multiplication test [21]). *Let M_1, M_2 be N-by-N symmetric matrices over Σ which agree on their diagonals. Suppose that $M_1 \neq M_2$. Then*

$$\Pr_{x,y \stackrel{R}{\leftarrow} \Sigma^N} [xM_1y \neq xM_2y] \geq \frac{3}{8}.$$

Furthermore, $\Pr_{x \stackrel{R}{\leftarrow} \Sigma^N} [xM_1 \neq xM_2] \geq 3/4$.

Proof. Let $M \stackrel{\mathrm{def}}{=} M_1 - M_2$. The probability that a uniformly selected combination of the rows of M yields an all-zero vector is 2^{-r}, where r is the rank of M. Since M is symmetric, not identically zero and has a zero diagonal, it must have rank at least 2. Thus, $\Pr_{x \stackrel{R}{\leftarrow} \Sigma^N} \left[xM \neq 0^N\right] \geq 3/4$ and the lemma follows. □

RMB DETECTORS. Suppose that A is actually linear. In that case, the following lemma provides a condition under which A respects the monomial basis. We start with a definition.

DEFINITION 3.17 (RMB detector). *Let A: $\mathcal{F}_l \to \Sigma$ and $f \in \mathcal{F}_l$. We say that f is a* detector for A *if*

$$\Pr_{g \stackrel{R}{\leftarrow} \mathcal{F}_l} [A(f' \cdot g) \neq 0] \geq 1/2,$$

where $f' = f$ if $A(f) = 0$ and $f' = f + \bar{1}$ otherwise.

The number of detectors is clearly related to the rejection probability of the RMB test. Suppose that A (or rather $\tilde{A}$) is linear. Clearly, if A respects the monomial basis, then it has no detectors. On the other hand, the following lemma asserts that if A does not respect the monomial basis, then it has many detectors.

LEMMA 3.18 (RMB test for linear functions). *Suppose $\tilde{A}\colon \mathcal{F}_l \to \Sigma$ is linear, $\tilde{A}(\chi_\emptyset) = 1$, and $\tilde{A}$ does not respect the monomial basis. Then at least a $3/4$ fraction of the functions in $\mathcal{F}_l$ are detectors for $\tilde{A}$.*

Proof. Let $N = 2^l$. We define a pair of N-by-N matrices whose rows and columns are indexed by the subsets of $[l]$. Specifically, for $S, T \subseteq [l]$, we set

$$M_1[S,T] = \tilde{A}(\chi_S) \cdot \tilde{A}(\chi_T),$$
$$M_2[S,T] = \tilde{A}(\chi_{S \cup T})\ .$$

Clearly, both M_1 and M_2 are symmetric, and they agree on the diagonal. Using $\tilde{A}(\chi_\emptyset) = 1$ we have, for every $T \subseteq [l]$,

$$M_1[\emptyset, T] = \tilde{A}(\chi_\emptyset) \cdot \tilde{A}(\chi_T) = 1 \cdot \tilde{A}(\chi_T) = M_2[\emptyset, T]. \tag{8}$$

By the hypothesis that $\tilde{A}$ does not respects the monomial basis, it follows that $M_1 \neq M_2$. Our aim is to relate the inequality of the above matrices to the existence of detectors for $\tilde{A}$. We first express the condition $\tilde{A}(fg) = \tilde{A}(f) \cdot \tilde{A}(g)$ in terms of these matrices.

Recall that $\mathcal{C}\colon \mathcal{F}_l \to \Sigma^{2^l}$ is the transformation which to any $f \in \mathcal{F}_l$ associates the vector $(\, C_f(S)\,)_{S \subseteq [l]}$ whose entries are the coefficients of f in its monomial series. Using the linearity of $\tilde{A}$ we note that

$$\begin{aligned}
\tilde{A}(f) \cdot \tilde{A}(g) &= \tilde{A}\left(\textstyle\sum_S C_f(S) \cdot \chi_S\right) \cdot \tilde{A}\left(\textstyle\sum_T C_g(T) \cdot \chi_T\right) \\
&= \left[\textstyle\sum_S C_f(S) \cdot \tilde{A}(\chi_S)\right] \cdot \left[\textstyle\sum_T C_g(T) \cdot \tilde{A}(\chi_T)\right] \\
&= \textstyle\sum_{S,T} C_f(S) \cdot \tilde{A}(\chi_S) \cdot \tilde{A}(\chi_T) \cdot C_g(T) \\
&= \mathcal{C}(f) M_1 \mathcal{C}(g)\ .
\end{aligned}$$

For the next step we first need the following.

Fact. Let $f, g \in \mathcal{F}_l$ and $U \subseteq [l]$. Then $C_{fg}(U) = \sum_{S \cup T = U} C_f(S) \cdot C_g(T)$.

Using this fact (and the linearity of $\tilde{A}$) we have

$$\begin{aligned}
\tilde{A}(fg) &= \tilde{A}\left(\textstyle\sum_U C_{fg}(U) \cdot \chi_U\right) \\
&= \textstyle\sum_U C_{fg}(U) \cdot \tilde{A}(\chi_U) \\
&= \textstyle\sum_U \sum_{S \cup T = U} C_f(S) \cdot C_g(T) \cdot \tilde{A}(\chi_U) \\
&= \textstyle\sum_{S,T} C_f(S) \cdot C_g(T) \cdot \tilde{A}(\chi_{S \cup T}) \\
&= \mathcal{C}(f) M_2 \mathcal{C}(g)\ .
\end{aligned}$$

Since $\tilde{A}$ is linear and $\tilde{A}(\bar{1}) = 1$ (as $\bar{1} = \chi_\emptyset$), we can rephrase the condition $A(f' \cdot g) \neq 0$, where $f' = f$ if $\tilde{A}(f) = 0$ and $f' = f + \bar{1}$ otherwise, as $A(f' \cdot g) \neq A(f') \cdot A(g)$. Thus, for every f (setting f' as above), we conclude that

$$A(f' \cdot g) \neq A(f') \cdot A(g) \quad \text{if and only if} \quad \mathcal{C}(f') M_2 \mathcal{C}(g) \neq \mathcal{C}(f') M_1 \mathcal{C}(g)\,.$$

A key observation is that $\mathcal{C}(f)$ and $\mathcal{C}(f')$ are identical in all entries except, possibly, for the entry corresponding to $\emptyset$ (i.e., $C_f(S) = C_{f'}(S)$ for all $S \neq \emptyset$). On the other hand, by Eq. (8), we have $M_1[\emptyset, \cdot] = M_2[\emptyset, \cdot]$. Thus,

$$A(f' \cdot g) \neq A(f') \cdot A(g) \quad \text{if and only if} \quad \mathcal{C}(f) M_2 \mathcal{C}(g) \neq \mathcal{C}(f) M_1 \mathcal{C}(g)\,.$$

Now we note that $\mathcal{C}$ is a bijection, so that if h is uniformly distributed in $\mathcal{F}_l$ then $\mathcal{C}(h)$ is uniformly distributed in Σ^{2^l}. Fixing any $f \in \mathcal{F}_l$ and setting f' as above, we

have, for $x = \mathcal{C}(f)$,

$$\Pr_{g \stackrel{R}{\leftarrow} \mathcal{F}_l}\left[\tilde{A}(f') \cdot \tilde{A}(g) \;=\; \tilde{A}(f'g)\right] = \Pr_{g \stackrel{R}{\leftarrow} \mathcal{F}_l}\left[\mathcal{C}(f)M_1\mathcal{C}(g) \;=\; \mathcal{C}(f)M_2\mathcal{C}(g)\right]$$
$$= \Pr_{y \stackrel{R}{\leftarrow} \Sigma^{2^l}}\left[xM_1y \;=\; xM_2y\right].$$

The latter probability is $1/2$ if $xM_1 \neq xM_2$ and zero otherwise. Invoking Lemma 3.16 we conclude that the first case, which coincides with f being a detector for $\tilde{A}$, holds for at least a $3/4$ fraction of the $f \in \mathcal{F}_l$. The lemma follows. □

Lemma 3.18 suggests that if we knew A was linear we could test that it respects the monomial basis by picking f, g at random and testing whether $A(f'g) = 0$, where $f' = f$ if $A(f) = 0$ and $f' = f + \bar{1}$ otherwise. The lemma asserts that in case A is linear and does not respect the monomial basis, we will have

$$\Pr_{f,g \stackrel{R}{\leftarrow} \mathcal{F}_l}\left[A(f'g) \neq 0\right] \geq \frac{3}{4} \cdot \frac{1}{2},$$

where $3/4$ is a lower bound on the probability that f is a detector for A and

$$\Pr_{g \stackrel{R}{\leftarrow} \mathcal{F}_l}\left[A(f'g) \neq 0\right] \geq \frac{1}{2}$$

for any detector f (by definition). However, we only know that A is close to linear. Still, we can perform an approximation of the above test via self-correction of the value $A(f'g)$. This, indeed, is our test as indicated in Figure 8.

THE RMB TEST. We are interested in lower bounding the probability $1 - \text{MBPASS}(A)$ that the test rejects, when f_1, f_2, f_3 are chosen at random, as a function of the distance of A to a linear function $\tilde{A}$, given that $\tilde{A}$ does not respect the monomial basis. We assume that A satisfies $A(f + \bar{1}) = A(f) + 1$ (for all $f \in \mathcal{F}_l$), as is the case in all our applications (since we use verifiers which access a $(\bar{1}, 1)$-folded function). The first item of the following lemma is in the spirit of previous analyses of analogous tests. The second item is somewhat unusual and will be used only in our construction of verifiers of free-bit complexity 2 (cf. section 5).

LEMMA 3.19 (RMB test—final analysis). *Let $A, \tilde{A}: \mathcal{F}_l \to \Sigma$ be functions such that $\tilde{A}$ linear but does* NOT *respect the monomial basis. Let $x = \text{Dist}(A, \tilde{A})$. Suppose that the function A satisfies $A(f + \bar{1}) = A(f) + 1$, for all $f \in \mathcal{F}_l$. Then*

(1) $1 - \text{MBPASS}(A) \;\geq\; \Gamma_{\text{RMB}}(x) \stackrel{\text{def}}{=} \frac{3}{8} \cdot (1 - 2x)$.

(2) $\Pr_{f_1,f_3 \stackrel{R}{\leftarrow} \mathcal{F}_l}\left[\exists f_2 \in \mathcal{F}_l \;\; s.t. \;\; \mathbf{MBTest}(A; f_1, f_2, f_3) = 1\right] \;\geq\; 2 \cdot \Gamma_{\text{RMB}}(x)$.

In particular, the lemma holds for $A_{(h,0),(\bar{1},1)}$, where $A: \mathcal{F}_l \to \Sigma$ is arbitrary and $h \in \mathcal{F}_l$. We will consider the linear function closest to $A_{(h,0),(\bar{1},1)}$, denoted $\tilde{A}$, and the case in which $\tilde{A}$ DOES NOT respect the monomial basis. (In this case $\text{Dist}(A_{(h,0),(\bar{1},1)}, \tilde{A})) = \text{Dist}(A_{(h,0),(\bar{1},1)}, \text{LIN}) \leq 1/2$.)

Proof. As a preparation for using Lemma 3.18, we first show that $\tilde{A}(\bar{1}) = 1$. For $x < 1/2$ this is justified by Corollary 3.14 (using the hypothesis $A(f + \bar{1}) = A(f) + 1$, $\forall f \in \mathcal{F}_l$). Otherwise (i.e., in case $x \geq 1/2$) the claimed lower bound (i.e., $\frac{3}{8} \cdot (1 - 2x) \leq 0$) holds vacuously.

Using Lemmas 3.18 and 3.13 we lower bound the rejection probability of the test, $1 - \text{MBPASS}(A)$, as follows:

$$\Pr_{f_1 \stackrel{R}{\leftarrow} \mathcal{F}_l}\left[f_1 \text{ is a detector for } \tilde{A}\right]$$
$$\cdot \min_{f \text{ is a } \tilde{A}\text{-detector}} \left\{\Pr_{f_2,f_3 \stackrel{R}{\leftarrow} \mathcal{F}_l}\left[\mathbf{MBTest}(A; f, f_2, f_3) = 1\right]\right\}$$

$$\geq \frac{3}{4} \cdot \min_{f \text{ is a } \tilde{A}\text{-detector}} \left\{ \Pr_{f_2, f_3 \stackrel{R}{\leftarrow} \mathcal{F}_l} [A(f'f_2 + f_3) \neq A(f_3)] \right\}$$

$$\geq \frac{3}{4} \cdot \min_{f \text{ is a } \tilde{A}\text{-detector}} \left\{ \Pr_{f_2, f_3 \stackrel{R}{\leftarrow} \mathcal{F}_l} \left[0 \neq \tilde{A}(f'f_2) \;=\; A(f'f_2 + f_3) - A(f_3) \right] \right\}$$

$$\geq \frac{3}{4} \cdot \frac{1}{2} \cdot \min_{f',\, g \quad \text{s.t.} \quad \tilde{A}(f'g) \neq 0} \left\{ \Pr_{f_3 \stackrel{R}{\leftarrow} \mathcal{F}_l} \left[\tilde{A}(f' \cdot g) \;=\; A(f' \cdot g + f_3) - A(f_3) \right] \right\}$$

$$\geq \frac{3}{8} \cdot (1 - 2x),$$

where the first inequality uses Lemma 3.18, the third inequality follows by the definition of a detector for $\tilde{A}$ (by which $\Pr_{g \stackrel{R}{\leftarrow} \mathcal{F}_l} \left[\tilde{A}(f'g) \neq 0 \right] \geq 1/2$), and the last inequality follows by Lemma 3.13. This concludes the proof of part 1. Part 2 is proven analogously with the exception that we don't lose a factor of two in the third inequality (since here f_2 is not selected at random but rather set existentially). □

Remark 3.20. An RMB test for arbitrary A's (rather than ones satisfying $A(f + \bar{1}) = A(f) + 1$, $\forall f \in \mathcal{F}_l$) can be derived by augmenting the above test with a test of $A(f + \bar{1}) = A(f) + 1$ for uniformly chosen $f \in \mathcal{F}_l$. The analysis of the augmented part is as in the circuit test (below).

3.5.3. Atomic projection test. The final test checks that the second function A_1 is not too far from the evaluation operator E_{a_1}, where $a_1 = \sigma(a)$ is a function of the string a whose evaluation operator is close to A. Here, unlike previous works (for instance, [23]), σ may be an arbitrary mapping from Σ^l to Σ^{l_1} rather than being a projection (i.e., satisfying $\sigma(x) = x^{(i_1)}, \ldots, x^{(i_{l_1})}$ for some sequence $1 \leq i_1 < \cdots < i_{l_1} \leq l$ and all $x \in \Sigma^l$). Thus, the term "projection test" is adopted for merely historical reasons.

LEMMA 3.21. *Let $A\colon \mathcal{F}_l \to \Sigma$ and let $\sigma\colon \Sigma^l \to \Sigma^{l_1}$ be a function. Let $a \in \Sigma^l$ and let $x = \mathrm{Dist}(A, E_a)$. Let $a_1 = \sigma(a) \in \Sigma^{l_1}$. Then $1 - \textsc{ProjPass}_\sigma(A, A_1) \geq \mathrm{Dist}(A_1, E_{a_1}) \cdot (1 - 2x)$.*

Proof. We lower bound the rejection probability as follows:

$$\begin{aligned}
&\Pr_{f \stackrel{R}{\leftarrow} \mathcal{F}_l \,;\, g \stackrel{R}{\leftarrow} \mathcal{F}_{l_1}} [A_1(g) \neq A(g \circ \sigma + f) - A(f)] \\
&\geq \Pr_{f \stackrel{R}{\leftarrow} \mathcal{F}_l \,;\, g \stackrel{R}{\leftarrow} \mathcal{F}_{l_1}} [A_1(g) \neq E_a(g \circ \sigma) \text{ and } A(g \circ \sigma + f) - A(f) = E_a(g \circ \sigma)] \\
&\geq \Pr_{g \stackrel{R}{\leftarrow} \mathcal{F}_{l_1}} [A_1(g) \neq E_a(g \circ \sigma)] \cdot (1 - 2x) \,.
\end{aligned}$$

Here we used Lemma 3.13 in the last step. Now we note that $E_a(g \circ \sigma) = (g \circ \sigma)(a) = g(\sigma(a)) = E_{a_1}(g)$. Hence the first term in the above product is just

$$\Pr_{g \stackrel{R}{\leftarrow} \mathcal{F}_{l_1}} [A_1(g) \neq E_{a_1}(g)] \;=\; \mathrm{Dist}(A_1, E_{a_1}) \,.$$

This concludes the proof. □

3.5.4. Atomic circuit test. For sake of elegance, we also present an atomic circuit test, denoted $\mathbf{CircTest}_h(A; f)$. The test consists of checking whether $A(h + f) = A(f)$, and it outputs 0 if equality holds and 1 otherwise. Assuming that A is close to some evaluation operator E_a, the atomic circuit test uses self-correction [27] to test that a given function h has value 0 at a. As explained above, this test is not needed since all of our proof systems will use a $(h, 0)$-folding (of A) and thus will

The MaxSNP inner verifier. Given functions $h \in \mathcal{F}_l$ and $\sigma\colon \Sigma^l \to \Sigma^{l_1}$, the verifier has access to oracles for $A\colon \mathcal{F}_l \to \Sigma$ and $A_1\colon \mathcal{F}_{l_1} \to \Sigma$. In addition it takes three $[0,1]$ valued parameters p_1, p_2, and p_3 such that $p_1+p_2+p_3 = 1$.

Pick $p \overset{R}{\leftarrow} [0,1]$.

Case: $p \le p_1$:
Pick $f_1, f_2 \overset{R}{\leftarrow} \mathcal{F}_l$.
LinTest$(A_{(h,0),(\bar{1},1)}; f_1, f_2)$.

Case: $p_1 < p \le p_1 + p_2$:
Pick $f_1, f_2, f_3 \overset{R}{\leftarrow} \mathcal{F}_l$.
MBTest$(A_{(h,0),(\bar{1},1)}; f_1, f_2, f_3)$.

Case: $p_1 + p_2 < p$:
Pick $f \overset{R}{\leftarrow} \mathcal{F}_l$ and $g \overset{R}{\leftarrow} \mathcal{F}_{l_1}$.
ProjTest$_\sigma(A_{(h,0),(\bar{1},1)}, A_1; f, g)$.

Remark: access to $A_{(h,0),(\bar{1},1)}(f)$ is implemented by accessing either $A(f)$, $A(f+h)$, $A(f+\bar{1})$, or $A(f+h+\bar{1})$.

FIG. 9. *The MaxSNP inner verifier* V_{SNPinner}.

impose $h(a) = 0$. The analysis lower bounds the rejection probability, as a function of the distance of A from linear, given that $h(a) = 1$.

LEMMA 3.22. *Let* $A\colon \mathcal{F}_l \to \Sigma$ *and let* $a \in \Sigma^l$. *Let* $h \in \mathcal{F}_l$ *and* $x = \text{Dist}(A, E_a)$. *If* $h(a) = 1$, *then* $1 - \textsc{CircPass}_h(A) \ge 1 - 2x$, *where*

$$\textsc{CircPass}_h(A) \overset{\text{def}}{=} \Pr_{f \overset{R}{\leftarrow} \mathcal{F}_l}[\,\textbf{CircTest}_h(A; f) = 0\,].$$

4. A new 3-query PCP and improved MaxSNP hardness results.

4.1. The MAXSNP verifier.

In this section we present a simple verifier which performs one of two simple checks, each depending on only three queries. This verifier will be the basis for the nonapproximability results for several MaxSNP problems, in particular Max3SAT, Max2SAT, and MaxCUT, whence its name.

4.1.1. The inner verifier. Figure 9 describes an inner verifier. Our verifier is adaptive, that is, some of its queries are determined as a function of answers to previous queries. (The adaptivity is not obvious from Figure 9; it is rather "hidden" in the RMB test; see section 3.5.2.) Adaptivity is used to improve the performance of our verifier and to strengthen the nonapproximability results which follow (cf. previous versions of this paper [20]).

The inner verifier, V_{SNPinner}, takes the usual length parameters l, l_1 as well as additional (probability) parameters p_1, p_2, and p_3 such that $p_1 + p_2 + p_3 = 1$. It performs just one test: with probability p_1 the linearity test; with probability p_2 the respect of monomial basis test; and with probability p_3 the projection test. Formally, this is achieved by picking p at random and making cases based on its value.[6] To

[6]For simplicity, p is depicted as being chosen as a random real number between 0 and 1. Of course we cannot quite do this, but we will see later that the values of p_1, p_2, p_3 in our final verifiers are appropriate constants. So in fact an appropriate choice of p can be made using $O(1)$ randomness, which is what we will implicitly assume.

improve the results, we perform the tests on a folding of A over both $(h,0)$ and $(\bar{1},1)$ (i.e., on $A_{(h,0),(\bar{1},1)}$). We stress that $A_{(h,0),(\bar{1},1)}$ is a virtual oracle which is implemented by the verifier which accesses the actual oracle A (on points determined by the definition of folding). We now examine the goodness of V_{SNPinner}. Recall the definitions of $\Gamma_{\text{lin}}(x)$ (specifically, note that $\Gamma_{\text{lin}}(x) \geq x$) and of $\Gamma_{\text{RMB}}(x) = \frac{3}{8}(1-2x)$, for all x.

Informally, the following lemma considers all the possible strategies of a "dishonest" prover and indicates the probability (denoted $1-\rho$) with which the verifier detects an error (when run against such strategies). The three cases correspond to the events that

(1) the function $A_{(h,0),(\bar{1},1)}$ may be very far from being linear;
(2) the function $A_{(h,0),(\bar{1},1)}$ is x-close to linear, for some $x < 1/2 - \delta_1$, but is not x-close to a valid codeword (i.e., to a linear function which respects the monomial basis); and
(3) the function $A_{(h,0),(\bar{1},1)}$ is x-close to a codeword, but the encoding of

$$\sigma(E^{-1}(A_{(h,0),(\bar{1},1)}))$$

is very far from the function A_1.

LEMMA 4.1 (soundness of V_{SNPinner}). *Suppose $\delta_1, \delta_2 > 0$, and $l, l_1 \in \mathcal{Z}^+$. Suppose $p_1, p_2, p_3 \in [0,1]$ satisfy $p_1 + p_2 + p_3 = 1$. Then the (l,l_1)-canonical inner verifier V_{SNPinner} is $(\rho, \delta_1, \delta_2)$-good, where $1-\rho = \min(T_1, T_2, T_3)$ and*

(1) $T_1 \stackrel{\text{def}}{=} p_1 \cdot (1/2 - \delta_1)$,
(2) $T_2 \stackrel{\text{def}}{=} \min_{x \leq 1/2-\delta_1} [\, p_1 \cdot \Gamma_{\text{lin}}(x) + p_2 \cdot \Gamma_{\text{RMB}}(x) \,]$,
(3) $T_3 \stackrel{\text{def}}{=} \min_{x \leq 1/2-\delta_1} [\, p_1 \cdot \Gamma_{\text{lin}}(x) + p_3 \cdot (1/2 - \delta_2)(1-2x) \,]$.

Proof. We consider an arbitrary pair of oracles, (A, A_1), and the behavior of V_{SNPinner} when given access to this pair of oracles. Our analysis is broken up into cases depending on (A, A_1); specifically, the first case-partition depends on the distance of $A_{(h,0),(\bar{1},1)}$ (i.e., the folding of A) from linear functions. We show that, in each case, either the verifier rejects with probability bounded below by one of the three quantities (above), or the oracle pair is such that rejection is not required.

Let $x = \text{Dist}(A_{(h,0),(\bar{1},1)}, \text{LIN})$.

Case 1. $x \geq 1/2 - \delta_1$. Lemma 3.15 implies that $1 - \text{LINPASS}(A_{(h,0),(\bar{1},1)}) \geq \Gamma_{\text{lin}}(x) \geq x \geq 1/2 - \delta_1$. (The second inequality follows from the fact that $\Gamma_{\text{lin}}(x) \geq x$ for all x.) Since V_{SNPinner} performs the atomic linearity test with probability p_1, we have

$$1 - \texttt{ACC}[\, V^{A,A_1}_{\text{SNPinner}}(\sigma, h) \,] \;\geq\; p_1 \cdot \left(\frac{1}{2} - \delta_1\right) \;\geq\; 1 - \rho. \tag{9}$$

Case 2. $x \leq 1/2 - \delta_1$. Lemma 3.15 implies that $1 - \text{LINPASS}(A_{(h,0),(\bar{1},1)}) \geq \Gamma_{\text{lin}}(x)$, and so the probability that V_{SNPinner} performs the linearity test and rejects is at least $p_1 \cdot \Gamma_{\text{lin}}(x)$. Now let $\tilde{A}$ be a linear function such that $\text{Dist}(A_{(h,0),(\bar{1},1)}, \tilde{A}) = x$. We consider the following subcases.

Case 2.1. $\tilde{A}$ does not respect the monomial basis. In this case, part 1 of Lemma 3.19 implies that $1 - \text{MBPASS}(A_{(h,0),(\bar{1},1)}) \geq \Gamma_{\text{RMB}}(x)$. So the probability that V_{SNPinner} performs the atomic respect of monomial basis test and rejects is at least $p_2 \cdot \Gamma_{\text{RMB}}(x)$. Since the event that the verifier performs a linearity test and the event that it performs a respect of monomial basis test are mutually exclusive, we can add the probabilities of rejection and thus get

$$1 - \texttt{ACC}[\, V^{A,A_1}_{\text{SNPinner}}(\sigma, h) \,] \;\geq\; p_1 \cdot \Gamma_{\text{lin}}(x) + p_2 \cdot \Gamma_{\text{RMB}}(x) \;\geq\; 1 - \rho. \tag{10}$$

Case 2.2. $\tilde{A}$ respects the monomial basis. By Proposition 3.2, $\tilde{A}$ is an evaluation operator. So there exists $a \in \Sigma^l$ such that $\tilde{A} = E_a$. So $\mathrm{Dist}(A_{(h,0),(\bar{1},1)}, E_a) = x$. Let $a_1 = \sigma(a)$. The proof splits into two further subcases.

Case 2.2.1. $d \stackrel{\text{def}}{=} \mathrm{Dist}(A_1, E_{a_1}) \geq 1/2 - \delta_2$. By Lemma 3.21 we have

$$1 - \textsc{ProjPass}_\sigma(A_{(h,0),(\bar{1},1)}, A_1) \geq d \cdot (1-2x) \geq \left(\frac{1}{2} - \delta_2\right) \cdot (1-2x) \; .$$

So the probability that V_{SNPinner} performs the projection test and rejects is at least $p_3 \cdot (1/2 - \delta_2)(1-2x)$. Thus, adding probabilities, as in Case 2.1, we get

$$(11) \quad 1 - \mathtt{ACC}\left[\, V^{A,A_1}_{\mathrm{SNPinner}}(\sigma, h)\,\right] \;\geq\; p_1 \cdot \Gamma_{\mathrm{lin}}(x) + p_3 \cdot (1/2 - \delta_2)(1-2x) \;\geq\; 1 - \rho.$$

Case 2.2.2. Else, we have $x = \mathrm{Dist}(A_{(h,0),(\bar{1},1)}, E_a) \leq 1/2 - \delta_1$ and $\mathrm{Dist}(A_1, E_{a_1}) < 1/2 - \delta_2$. Thus the functions $A_{(h,0),(\bar{1},1)}$ and A_1 satisfy conditions (2.1) and (2.2) in Definition 3.9.

Observe that the only case which does not yield $1 - \mathtt{ACC}\left[\, V^{A,A_1}_{\mathrm{PCPinner}}(\sigma, h)\,\right] \geq 1 - \rho$ is Case (2.2.2). However, Case (2.2.2) satisfies conditions (2.1) and (2.2) of Definition 3.9. Thus, V_{PCPinner} satisfies condition (2) of Definition 3.9. Clearly, V_{PCPinner} also satisfies condition (1) of Definition 3.9, and thus the lemma follows. □

The upper bound on the soundness error of V_{SNPinner}, provided by Lemma 4.1, is somewhat complicated to grasp. Fortunately, using $\Gamma_{\mathrm{RMB}}(x) = \frac{3}{8}(1-2x)$ and $\Gamma_{\mathrm{lin}}(x) \geq x$, for all $x \leq 1/2$, we can simplify the expression as follows.

Claim 4.2. *Let T_1, T_2, and T_3 be as in Lemma* 4.1, $\delta = \max(\delta_1, \delta_2) > 0$, *and $p_1, p_2, p_3 \in [0,1]$ satisfy $p_1 + p_2 + p_3 = 1$. Then, $T_2 \geq \min\{\frac{1}{2}p_1, \frac{3}{8}p_2\}$, $T_3 \geq \min\{\frac{1}{2}p_1, \frac{1}{2}p_3\} - \delta$, and*

$$\min\{T_1, T_2, T_3\} \geq \min\left\{\frac{1}{2}p_1, \frac{3}{8}p_2, \frac{1}{2}p_3\right\} - \delta \; .$$

Interestingly, this lower bound is tight.

Proof. Clearly, $T_1 = (\frac{1}{2} - \delta_1)p_1 \geq \frac{1}{2}p_1 - \delta$. To analyze T_2, let $h(x) \stackrel{\text{def}}{=} p_1 \cdot \Gamma_{\mathrm{lin}}(x) + p_2 \cdot \Gamma_{\mathrm{RMB}}(x)$.

Fact 1. $\min_{x \leq 1/2}\{h(x)\} = \min\{\frac{3}{8}p_2, \frac{1}{2}p_1\} = \min\{h(0), h(1/2)\}$.

Proof. By considering two cases and using $\Gamma_{\mathrm{lin}}(x) \geq x$ and $\Gamma_{\mathrm{RMB}}(x) = \frac{3}{8} - \frac{3}{4}x$.

Case 1. $p_1 \geq \frac{3}{4}p_2$;

$$h(x) \geq p_1 x + \frac{3}{8}p_2 - \frac{3}{4}p_2 x = \frac{3}{8}p_2 + \left(p_1 - \frac{3}{4}p_2\right) \cdot x \geq \frac{3}{8}p_2.$$

Case 2. $p_1 \leq \frac{3}{4}p_2$;

$$h(x) \geq p_1 x + \frac{3}{8}p_2 - \frac{3}{4}p_2 x = \frac{1}{2}p_1 + \left(\frac{3}{4}p_2 - p_1\right) \cdot \left(\frac{1}{2} - x\right) \geq \frac{1}{2}p_1.$$

The fact follows by observing that $h(0) = \frac{3}{8}p_2$ and $h(1/2) = \frac{1}{2}p_1$. □

Thus, we have $T_2 = \min_{x \leq 1/2 - \delta_1}[h(x)] \geq \min\{\frac{1}{2}p_1, \frac{3}{8}p_2\}$. The term T_3 is analyzed similarly, by defining $g(x) \stackrel{\text{def}}{=} p_1 \cdot \Gamma_{\mathrm{lin}}(x) + p_3 \cdot (1-2x)/2$, and using the following fact.

Fact 2. $\min_{x \leq 1/2}\{g(x)\} = \min\{\frac{1}{2}p_3, \frac{1}{2}p_1\} = \min\{g(0), g(1/2)\}$.

Proof. The proof is by considering two cases and using $\Gamma_{\mathrm{lin}}(x) \geq x$.

Case 1. $p_1 \geq p_3$;

$$g(x) \geq p_1 x + \frac{1}{2}p_3 - p_3 x = \frac{1}{2}p_3 + (p_1 - p_3) \cdot x \geq \frac{1}{2}p_3.$$

Case 2. $p_1 \leq p_3$;

$$g(x) \geq p_1 x + \frac{1}{2}p_3 - p_3 x = \frac{1}{2}p_1 + (p_3 - p_1) \cdot \left(\frac{1}{2} - x\right) \geq \frac{1}{2}p_1.$$

The fact follows by observing that $g(0) = \frac{1}{2}p_3$ and $g(1/2) = \frac{1}{2}p_1$. □

Thus, we have $T_3 \geq \min_{x \leq 1/2 - \delta_2}[g(x)] - \delta \geq \min\{\frac{1}{2}p_1, \frac{1}{2}p_3\} - \delta$. The claim follows. □

4.1.2. Main application: The MaxSNP verifier. We are now ready to state the main result of this section. It is a simple verifier for NP which achieves soundness error approaching 85% while performing one of two very simple tests.

PROPOSITION 4.3 (the MaxSNP verifier). *For any $\gamma > 0$ and for any language $L \in$ NP, there exists a verifier V_{SNP} for L such that*

- *V_{SNP} uses logarithmic randomness and has perfect completeness;*
- *V_{SNP} has soundness error $\frac{17}{20} + \gamma$; and*
- *on access to an oracle π (and according to the outcome of the verifier's coin tosses), the verifier V_{SNP} performs one of the following actions:*
  (1) Parity check. *V_{SNP} determines a bit b, makes three queries q_1, q_2, and q_3, and rejects if $\pi(q_1) \oplus \pi(q_2) \oplus \pi(q_3) \neq b$.*
  (2) RMB check. *V_{SNP} determines two bits b_0, b_1, makes three out of four predetermined queries q_1, q_2, q_3, and q_4, and rejects if either $(\pi(q_1) = 0) \wedge (\pi(q_2) \oplus \pi(q_4) \neq b_0)$ or $(\pi(q_1) = 1) \wedge (\pi(q_3) \oplus \pi(q_4) \neq b_1)$.*
  That is, the verifier inspects $\pi(q_1)$ and consequently checks either $\pi(q_2) \oplus \pi(q_4) \stackrel{?}{=} b_0$ or $\pi(q_3) \oplus \pi(q_4) \stackrel{?}{=} b_1$.

  Furthermore, the probability (over its coin tosses) that V_{SNP} performs a parity check is $\frac{3}{5}$ (and the probability that V_{SNP} performs an RMB check is $\frac{2}{5}$).

Proof. Set $\delta_1 = \delta_2 = \gamma/2$ and $\epsilon = \frac{\gamma}{2} \cdot (16\delta_1^2\delta_2^2) = \frac{\gamma^5}{2} > 0$. Now, let l and l_1 be integers such that the outer verifier, V_{outer}, guaranteed by Lemma 3.8 is (l, l_1)-canonical and ϵ-good for L. Consider the (l, l_1)-canonical inner verifier V_{SNPinner}, working with the parameters p_1, p_2, and p_3 set to minimize its error. Obviously this calls for setting $1/2p_1 = \frac{3}{8}p_2 = 1/2p_3$, which yields

$$p_1 = \frac{3}{10} \;;\; p_2 = \frac{4}{10} \;;\; p_3 = \frac{3}{10}. \tag{12}$$

Let V_{SNP} be the verifier obtained by composing V_{outer} with V_{SNPinner}.

We start by analyzing the soundness error of V_{SNP}. By Lemma 4.1 and Claim 4.2, we know that the inner verifier V_{SNPinner}, with p_i's as in Eq. (12), is $(\rho, \delta_1, \delta_2)$-good, for

$$\rho \;\leq\; 1 - \frac{1}{2} \cdot p_3 + \delta_1 \;=\; 1 - \frac{3}{20} + \frac{1}{2} \cdot \gamma\,.$$

Invoking Theorem 3.12, we upper bound the soundness error of V_{SNP} by $1 - \frac{3}{20} + \frac{\gamma}{2} + \frac{\epsilon}{16\delta_1^2\delta_2^2}$ which by the setting of ϵ yields the claimed soundness bound (of $0.85 + \gamma$).

Clearly, V_{SNP} uses logarithmic randomness, has perfect completeness, and its computation on the answers of the oracles are determined by V_{SNPinner}. It is left to

observe that each of the three tests (i.e., linearity, monomial basis, and projection), performed by V_{SNPinner}, is either a parity check or an RMB check and that the latter occurs with probability 0.4. First observe that with probability p_1, V_{SNPinner} performs $\mathbf{LinTest}(A_{(h,0),(\bar{1},1)}; f_1, f_2)$, where $f_1, f_2 \in \mathcal{F}_l$. Recall that query f to $A_{(h,0),(\bar{1},1)}$ translates to a query in the set $\{f, f+h, f+\bar{1}, f+h+\bar{1}\}$ answered by A and that the answer is possibly complemented (by adding 1 mod 2). Thus, the above linearity test translates to checking the exclusive-or of three predetermined locations of A against a predetermined bit b (i.e., this bit is determined by the number of times which have shifted a potential query by $\bar{1}$). Similarly, $\mathbf{MBTest}(A_{(h,0),(\bar{1},1)}; f_1, f_2, f_3)$ translates to an RMB check with b_0, b_1, and the four possible queries being determined by the folding over $\bar{1}$. Finally, we observe that the projection test, performed by V_{SNPinner}, also amounts to a parity check, this time on answers taken from two different oracles (which can actually be viewed as one oracle). □

Remark 4.4 (a tedious one). The probability that verifier V_{SNP}, of the above proposition, makes two identical queries is negligible. Specifically, it can be made smaller than γ (mentioned in the proposition). Thus, we can ignore this case[7] in the next two sections and assume, without loss of generality, that all queries are distinct.

Implementing the MaxSNP verifier via gadgets. In the following sections we use the verifier of Proposition 4.3 to obtain hardness results for various variants of MaxSAT as well as for MaxCUT. The hardness results are obtained by constructing an instance of the problem at hand so that the instance represent the verifier's computation on input x. The primary aspect of the reduction is the construction of gadgets which reflect the result of the verifier's computation (i.e., accept/reject) after performing one of the two types of checks, i.e., parity check or RMB check. We define a performance measure of a gadget and relate the hardness result achieved to the performance measure obtained by the gadgets in use.[8]

Sources of our improvements. The explicit statement of a generic verifier for deriving MaxSNP hardness results is a novelty of our paper. Thus, a quantitative comparison to previous works is not readily available. Certainly, we improve over these works thanks to the use of the new long code-based inner verifier, the atomic tests and their analysis in section 3.5, the new idea of folding, and the improved analysis of linearity testing due to [17].

4.1.3. Another application: Minimizing soundness error in 3-query pcp. As a direct corollary to Proposition 4.3, we obtain the following.

Theorem 4.5. *For any* $s > 0.85$, $\text{NP} \subseteq \text{PCP}_{1,s}[\,\mathsf{coins} = \log; \mathsf{query} = 3; \mathsf{free} = 2\,]$.

4.2. Satisfiability problems. In this section we mainly deal with CNF formulae. However, the last subsection deals with formulae consisting of a conjunction of PARITY (rather than OR) clauses. Refer to section 2.4 for definitions, in particular for the problem MaxXSAT and the promise problem Gap-XSAT. Recall that $\text{MaxSAT}(\varphi)$ is the maximum number of simultaneously satisfiable clauses in

[7] Formally, suppose that, when it occurs, the verifier performs some standard check on fixed different queries. This modification increases the soundness error by at most γ which tends to zero anyhow.

[8] Given that the performance of the various gadgets might be different for the different checks, one might suspect that it might have been a better idea to first construct the gadgets and then to optimize the soundness of V_{SNP}, keeping in mind the relative performance measures of the two kinds of gadgets being employed. Surprisingly enough it turns out (cf. [20]) that the optimization is not a function of the performance of the gadgets and indeed the choice of parameters p_1, p_2, and p_3, as in Eq. (12), is optimal for the following reductions.

formula φ, and $\overline{\text{MaxSAT}}(\varphi) = \text{MaxSAT}(\varphi)/\|\varphi\|$ is the normalized version, where $\|\varphi\|$ is the number of clauses in formula φ. See section 2.4.3 for a description of previous work.

A consequence of the following theorem (obtained by applying Proposition 2.5) is that, assuming P $\neq$ NP, there is no polynomial-time algorithm to approximate the following: (1) Max3SAT within a factor of 1.038; (2) MaxE3SAT within a factor of 1.038; (3) Max2SAT within a factor of 1.013.

THEOREM 4.6 (MaxSAT nonapproximability results). *The following problems are* NP*-hard:*

(1) Gap-$3SAT_{c,s}$ *with* $c = 1$ *and* $s = 26/27$.
(2) Gap-$E3SAT_{c,s}$ *with* $c = 1$ *and* $s = 26/27$.
(3) Gap-$2SAT_{c,s}$ *for some* $0 < s < c < 1$ *satisfying* $c > 0.9$ *and* $c/s = 74/73$.

Actually, items 1 and 2 hold for any $s > 1 - \frac{3}{80}$ whereas item 3 holds as long as $\frac{c}{s} < 1 + \frac{3}{217}$. Item 1 is implied by item 2 so we will prove only the latter.

4.2.1. The hardness of MaxE3SAT and Max2SAT.

GADGETS. In the context of MaxSAT problems, we may easily replace a condition of the form $a + b + c = 1$ by $\bar{a} + b + c = 0$, where $\bar{a}$ is the negation of the variable a. Thus, the task of designing gadgets is simplified, and we need to implement two (simplified) types of checks: the parity check (checking that $a + b = c$ for a, b, and c obtained from the oracle) and the RMB check for a, b_0, b_1, and c obtained from the oracle). Accordingly a parity check (PC) gadget, $\text{PC}(a, b, c, x_1, x_2, \ldots, x_n)$, is a set of clauses over three *distinguished* variables a, b, c and n auxiliary variables $x_1, \ldots, x_n$. It is an (α, β)*-PC gadget* if the following is true: if $a + b = c$, then $\text{MaxSAT}(\text{PC}(a, b, c, x_1, x_2, \ldots, x_n)) = \alpha$; else it is at most $\alpha - \beta$. Similarly a respect-monomial-basis check (RMBC) gadget, $\text{RMBC}(a, b_0, b_1, c, x_1, \ldots, x_n)$, is a set of clauses over four *distinguished* variables a, b_0, b_1, c and n auxiliary variables $x_1, \ldots, x_n$. It is an (α, β)*-RMBC gadget* if the following is true: if $b_a = c$, then $\text{MaxSAT}(\text{RMBC}(a, b_0, b_1, c, x_1, x_2, \ldots, x_n)) = \alpha$; else it is at most $\alpha - \beta$. We stress that in both cases the maximum number of clauses which are simultaneously satisfied is at most α. A gadget is said to be an X-SAT gadget if, as a formula, it is an X-SAT formula.

The following lemma describes how gadgets of the above form can be used to obtain the hardness of MaxSAT.

LEMMA 4.7 (MaxSAT implementation of a verifier). *Let V be a verifier for L of logarithmic randomness, with perfect completeness and soundness s, such that V performs either a single parity check (with probability q) or a single RMB check (with probability $1 - q$). Furthermore, suppose that in either case, the verifier never makes two identical queries. If there exists an (α_1, β)-parity-check X-SAT gadget containing m_1 clauses and an (α_2, β)-RMBC X-SAT gadget containing m_2 clauses, then L reduces to* Gap-$XSAT_{c',s'}$ *for*

$$c' = \frac{\alpha_1 q + \alpha_2(1-q)}{m_1 q + m_2(1-q)},$$

$$s' = \frac{\alpha_1 q + \alpha_2(1-q) - (1-s)\beta}{m_1 q + m_2(1-q)}.$$

In particular $\frac{c'}{s'} \geq 1 + \frac{(1-s)\beta}{\alpha_1 q + \alpha_2(1-q) - (1-s)\beta}$.

Remark 4.8. In the above lemma, we have assumed that both the PC and RMBC gadgets have the same second parameter β. This assumption is not really a restriction since we can transform a pair of (α_1, β_1)-PC and (α_2, β_2)-RMBC gadgets into a pair of $(\alpha_1\beta_2, \beta_1\beta_2)$-PC and $(\alpha_2\beta_1, \beta_1\beta_2)$-RMBC gadgets, thereby achieving this feature. (Actually, what really matters are the fractions α_i/β.)

Proof. Let $\mathrm{PC}(a, b, c, x_1, \ldots, x_{n_1})$ be the parity check gadget and let $\mathrm{RMBC}(a, b, c, d, x_1, \ldots, x_{n_2})$ be the RMBC gadget. We encode V's computation on input x by a CNF formula φ_x. Corresponding to every bit $\pi[q]$ of the proof (oracle) accessed by the verifier V, we create a variable $y[q]$. In addition we create some auxiliary variables $y_{\mathrm{Aux}}[R, i]$ for each random string R used by the verifier V and i going from 1 to $\max(n_1, n_2)$. For each such R we will construct a formula φ_R which encodes the computation of the verifier when its coins are R. The union of all these formulae will be our φ_x.

On random string R, if the verifier performs a parity check on bits $\pi[q_1], \pi[q_2]$, and $\pi[q_3]$, then φ_R consists of the clauses $PC(y[q_1], y[q_2], y[q_3], y_{\mathrm{Aux}}[R, 1], \ldots, y_{\mathrm{Aux}}[R, n_1])$. On the other hand if the verifier performs an RMB check on bits $\pi[q_1], \pi[q_2], \pi[q_3]$, $\pi[q_4]$, then φ_R consists of the clauses $RMBC(y[q_1], y[q_2], y[q_3], y[q_4], y_{\mathrm{Aux}}[R, 1], \ldots, y_{\mathrm{Aux}}[R, n_2])$.

Let N denote the number of possible random strings used by V. Observe that the number of clauses in φ_x equals $m_1 \cdot qN + m_2 \cdot (1-q)N$. We now analyze the value of $\mathrm{MaxSAT}(\varphi_x)$.

If $x \in L$, then there exists an oracle π such that $V^\pi(x)$ always accepts. Consider the assignment $y[q] = \pi[q]$ (i.e., $y[q]$ is true iff $\pi[q] = 1$). Then for every R, there exists an assignment to the variables $y_{\mathrm{Aux}}[R, i]$ such that the number of clauses of φ_R that are satisfied by this assignment is α_1 if R corresponds to a parity check and α_2 if R corresponds to an RMB check. Since qN of the gadgets are PC gadgets and $(1-q)N$ of the gadgets are RMBC gadgets, we have $\mathrm{MaxSAT}(\varphi_x) \geq qN\alpha_1 + (1-q)N\alpha_2$, and the expression for c' follows.

Now consider the case when $x \notin L$. We prove below that if there exists an assignment which satisfies $qN\alpha_1 + (1-q)N\alpha_2 - (1-s)N\beta$ clauses of φ_x, then there exists an oracle π such that $V^\pi(x)$ accepts with probability at least s. Since we know this cannot happen, we conclude that $\mathrm{MaxSAT}(\varphi_x) < qN\alpha_1 + (1-q)N\alpha_2 - (1-s)N\beta = s'|\varphi_x|$.

To prove the above claim, we convert any assignment to the variables y into an oracle π in the natural way, i.e., $\pi[q] = 1$ iff $y[q]$ is true. Now, by the property of the gadgets, if a PC gadget $PC(y[q_1], y[q_2], y[q_3], y_{\mathrm{Aux}}[R, 1], \ldots)$ has more than $\alpha_1 - \beta$ clauses satisfied, then $\pi[q_1] \oplus \pi[q_2] = \pi[q_3]$. In turn this implies that the verifier V accepts π on random string R. A similar argument can be made about the random strings R which correspond to RMB checks. We also use the property that a PC (resp., RMB) gadget cannot have more than α_1 (resp., α_2) satisfied clauses, even if the claim it checks does hold. Thus, if an assignment satisfies $qN \cdot (\alpha_1 - \beta) + (1-q)N \cdot (\alpha_2 - \beta) + sN\beta$ clauses, then there must exist sN random strings R on which V accepts. This proves the claim and the lemma follows. □

Figure 10 describes gadgets which will be used for our MaxE3SAT construction; notice that they are *exact*-3-SAT gadgets. We have a $(4, 1)$-PC gadget, PC_3, and a $(4, 1)$-RMB gadget, RMBC_3, each consisting of four clauses in which all the clauses have exactly three variables. Both gadgets have no auxiliary variables. The $\mathrm{PC}_3(a, b, c)$ gadget is merely the canonical 3CNF of the expression $a + b + c = 0$. The first two clauses in the $\mathrm{RMBC}_3(a, b, b', c)$ gadget are the canonical 3CNF of the

The MaxE3SAT gadgets.

$$\mathrm{PC}_3(a,b,c) = \{(a \vee b \vee \overline{c}),\ (a \vee \overline{b} \vee c),\ (\overline{a} \vee b \vee c),\ (\overline{a} \vee \overline{b} \vee \overline{c})\}$$

$$\mathrm{RMBC}_3(a,b,b',c) = \{(a \vee b \vee \overline{c}),\ (a \vee \overline{b} \vee c),\ (\overline{a} \vee b' \vee \overline{c}),\ (\overline{a} \vee \overline{b'} \vee c),\ \}$$

FIG. 10. *The* Max*E*3SAT *gadgets.*

The MAX2SAT gadgets.

$$\begin{aligned}\mathrm{PC}_2(a,b,c,x_{00},x_{01},x_{10},x_{11}) = \\ \{&(\overline{x_{00}} \vee \overline{a}), (\overline{x_{00}} \vee \overline{b}), (x_{00} \vee c),\\ &(\overline{x_{01}} \vee a), (\overline{x_{01}} \vee \overline{b}), (x_{01} \vee \overline{c}),\\ &(\overline{x_{10}} \vee \overline{a}), (\overline{x_{10}} \vee b), (x_{10} \vee \overline{c}),\\ &(\overline{x_{11}} \vee a), (\overline{x_{11}} \vee b), (x_{11} \vee c)\}.\end{aligned}$$

$$\begin{aligned}\mathrm{RMBC}_2(a,b,b',c,x_{00},x_{11},y_{00},y_{11}) = \\ \{&(\overline{x_{00}} \vee \overline{b}), (\overline{x_{00}} \vee \overline{c}), (a \vee x_{00}),\\ &(\overline{x_{11}} \vee b), (\overline{x_{11}} \vee c), (a \vee x_{11}),\\ &(\overline{y_{00}} \vee \overline{b'}), (\overline{y_{00}} \vee \overline{c}), (\overline{a} \vee y_{00}),\\ &(\overline{y_{11}} \vee b'), (\overline{y_{11}} \vee c), (\overline{a} \vee y_{11})\}.\end{aligned}$$

FIG. 11. *The* Max2SAT *gadgets.*

expression $(a = 0) \Rightarrow (b = c)$, whereas the latter two clauses are the canonical 3CNF of the expression $(a = 1) \Rightarrow (b' = c)$. Figure 11 similarly describes 2-SAT gadgets for our Max2SAT construction. We have a $(11, 1)$-PC gadget, PC_2, and a $(11, 1)$-RMB gadget, RMBC_2, each consisting of 12 clauses. Each gadget has four auxiliary variables. The auxiliary variable $x_{\tau\sigma}$ in the PC_2 gadget is supposed to be the indicator of the event $((a = \sigma) \wedge (b = \tau))$. Thus, $a + b = c$ allows us to satisfy 11 clauses by appropriately setting the indicator variables (e.g., if $a = b = c = 0$, then setting $x_{00} = 1$ and the other $x_{\tau\sigma}$'s to 0 satisfies all clauses except the last one). The RMBC_2 gadget is composed of two parts; the first six clauses handle the expression $(a = 0) \Rightarrow (b = c)$, whereas the latter six clauses are for the expression $(a = 1) \Rightarrow (b' = c)$.

LEMMA 4.9 (SAT gadgets). *The following gadgets exist.*

(1) *E*3-SAT *gadgets: a* $(4, 1)$*-PC gadget of four clauses and a* $(4, 1)$*-RMB gadget of four clauses.*

(2) 2-SAT *gadgets: a* $(11, 1)$*-PC gadget of* 12 *clauses and a* $(11, 1)$*-RMB gadget of* 12 *clauses.*

Remark 4.10. In previous versions of this work [20], it was observed that a ratio of 4 between the number of clauses and the second parameter (i.e., β) is minimal for both E 3-SAT gadgets. Several questions regarding the α/β ratios achievable by 3-SAT and 2-SAT gadgets were posed. Answers were subsequently provided in [84], which undertakes a general study of the construction of optimal gadgets.

Proof of Lemma 4.9. We use the gadgets presented in Figures 10 and 11. The claim regarding E3-SAT follows from the motivating discussion above (i.e., by which these gadgets are merely the canonical 3CNF expressions for the corresponding conditions). The analysis of the 2-SAT gadgets in Figure 11 is straightforward but tedious; it is omitted from this version and can be found in previous versions of this work [20]. □

Proof of Theorem 4.6. The theorem follows by applying Lemma 4.7 to the verifier of Proposition 4.3 and the gadgets of Lemma 4.9. Details follow.

Recall that by Remark 4.4, we may assume that the verifier does not make two identical queries. Applying Lemma 4.7 to the verifier of Proposition 4.3 we obtain a reduction of any language in NP to $\mathsf{Gap\text{-}XSAT}_{c',s'}$ for values of c' and s' determined as a function of the gadget parameters, the probability parameter q, and the soundness s of the verifier of Proposition 4.3. Specifically, we observe that for E3-SAT we have $c' = 1$ (since $\alpha_i = m_i = 4$ for $i = 1, 2$), whereas for 2-SAT we have $0.9 < c' < 1$ (since $\frac{\alpha_i}{m_i} = \frac{11}{12} = 4$ for $i = 1, 2$). In both cases, $\beta = 1$ and the expression for c'/s' is given by

$$1 + \frac{1 - s}{q\alpha_1 + (1 - q)\alpha_2 - (1 - s)}, \tag{13}$$

where s and q are determined by Proposition 4.3; that is, (for every $\gamma > 0$),

$$s = 1 - \frac{3}{20} + \gamma, \tag{14}$$

$$q = \frac{3}{5}, \tag{15}$$

Substituting Eqs. (14) and (15) into Eq. (13), and letting $\gamma \to 0$, we get

$$\frac{c'}{s'} \to 1 + \frac{3}{12\alpha_1 + 8\alpha_2 - 3}.$$

The bounds for E3-SAT and 2-SAT now follow by using the α_i's values of Lemma 4.9. In particular, for E3-SAT we get $s' \to 77/80$ and for 2-SAT we get $\frac{c'}{s'} \to 1 + \frac{3}{217}$. □

We conclude this subsection by presenting a variant of Lemma 4.7. This variant refers only to 3-SAT formulae, but makes no restrictions on the verifier in the PCP system.

LEMMA 4.11 (Max3SAT implementation of a generic verifier). *Let L be in* $\mathrm{PCP}_{1,1-\delta}[\log, 3]$, *for some* $0 < \delta < 1$. *Then, L reduces to* $\mathsf{Gap\text{-}3SAT}_{1,1-\frac{\delta}{4}}$.

Proof. Let V be a verifier as guaranteed by the hypothesis. Building on Lemma 4.7, it suffices to show that the computation of V on any possible random tape can be captured by a 3CNF formula with at most four clauses. We consider the depth-3 branching program which describes the acceptance of V on a specific random tape. (The variables in this program correspond to queries that the verifier may make on this fixed random tape. Since the verifier may be adaptive, different variables may appear on different paths.) In case this tree has at most four rejecting leaves (i.e., marked FALSE), writing corresponding 3CNF clauses (which state that these paths are not followed), we are done. Otherwise, we consider the four depth-1 subtrees. For each such subtree we do the following. In case both leaves are marked FALSE, we write a 2CNF clause (which states that this subtree is not reached at all). In case a single leaf is marked FALSE, we write one 3CNF clause (as above), and if no leaf is marked FALSE, we write nothing. □

Remark 4.12. *The above argument can be easily extended to show that, for any* $0 \leq \epsilon, \delta < 1$,

$$\mathrm{PCP}_{1-\epsilon,1-\delta}[\log, 3] \leq_D^K \textsf{Gap-3SAT}_{1-\epsilon,1-\frac{\delta}{4}}.$$

4.2.2. Maximum satisfiable linear constraints (parity clauses). Analogously to the MaxSAT problems considered above, we consider parity clauses rather than disjunctive clauses. In other words, we are given a system of linear equations over GF(2), and need to determine the maximum number of equations which may be simultaneously satisfied. The problem in question is MaxLinEq (cf. section 2.4.2). Here we provide an explicit hardness factor via a direct reduction from the MaxSNP verifier.

THEOREM 4.13. $\textsf{Gap-MaxLinEq}_{c,s}$ *is* NP*-hard for* $c = 6/7$ *and any* $\frac{c}{s} < 8/7$.

Proof. The theorem follows by constructing appropriate gadgets. A $(1,1)$-PC gadget is straightforward here. We present a $(3,2)$-RMB gadget consisting of four equations. Specifically, for $\mathrm{RMB}(a, b_0, b_1, c)$ we present the equations $b_0 + c = 0$, $a + b_0 + c = 0$, $b_1 + c = 0$, and $a + b_1 + c = 1$. Observe that we can think of the RMB gadget as a $(1.5, 1)$ gadget with two clauses (or, equivalently, think of the parity gadget as a $(2,2)$ gadget with two clauses).

We obtain a hardness for $\textsf{Gap-MaxLinEq}_{c',s'}$ by proceeding as in the proof of Theorem 4.6, where

$$\frac{c'}{s'} \to 1 + \frac{3}{12\alpha_1 + 8\alpha_2 - 3} = 1 + \frac{3}{12 + 12 - 3} = \frac{8}{7},$$

and $c' = \frac{3\alpha_1 + 2\alpha_2}{3m_1 + 2m_2} = \frac{6}{7}$. ∎

4.3. MaxCUT. Refer to section 2.4 for the definition of the MaxCUT problem and the associated gap problem $\textsf{Gap-MaxCUT}_{c,s}$. See section 2.4.3 for discussion of status and previous work. The following theorem (combined with Proposition 2.5) shows that MaxCUT is NP-hard to approximate to within a factor of 1.014. We note that the result of the following theorem holds also when the weights of the graph are presented in unary (or, equivalently, when considering unweighted graphs with parallel edges).

THEOREM 4.14 (MaxCUT nonapproximability result). $\textsf{Gap-MaxCUT}_{c,s}$ *is NP-hard for some* c, s *satisfying* $c > 0.6$ *and* $c/s > 1.014$ *(anything below* $72/71$*).*

A weaker result can be obtained for simple graphs without weights or parallel edges. In particular, one may reduce the MaxCUT problem for graphs with parallel edges to MaxCUT for simple graphs by replacing every edge by a path of three edges. This causes a loss of a factor of 3 in the hardness factor, that is, we would get a hardness factor of $214/213$ for the MaxCUT problem restricted to simple graphs. A better reduction which preserves the nonapproximation ratio has been recently suggested by Crescenzi, Silvestri, and Trevisan [34].

GADGETS. Unlike with the MaxSAT problem, here we cannot negate variables at zero cost. Still, we first define simplified gadgets for parity and RMB checking and make the necessary adaptations inside Lemma 4.15.

Gadgets will be used to express the verifier's computation in terms of cuts in graphs. A parity check gadget $\text{PC-CUT}(a, b, c, T; x_1, \ldots, x_n)$ is a weighted graph on $n+4$ vertices. Of these three vertices, a, b, c correspond to oracle queries made by the verifier. The vertex T will be a special vertex mapping cuts to truth values so that a vertex corresponding to an oracle query is considered set to 1 if it resides in the T-side of the cut (i.e., a is considered set to 1 by a cut $(S, \overline{S})$ iff either $a, T \in S$ or $a, T \in \overline{S}$).

The gadget is an (α, β)-PC gadget if MaxCUT(PC-CUT$(a, b, c, T; x_1, \ldots, x_n)$) is exactly α when restricted to cuts which induce $a+b=c$ (i.e., either 0 or 2 of the vertices $\{a, b, c\}$ lie on the same side of the cut as T), and is at most $\alpha - \beta$ when restricted to cuts for which $a+b \neq c$. A cut gadget to check if $a+b \neq c$ can be defined similarly. Similarly, a weighted graph RMBC-CUT$(a, b_0, b_1, c, T; x_1, \ldots, x_n)$ is an (α, β)-RMBC gadget if it satisfies the property that MaxCUT(RMBC-CUT$(a, b_0, b_1, c, T; x_1, \ldots, x_n)$) is exactly α when restricted to cuts satisfying $b_a = c$ and is at most $\alpha - \beta$ otherwise. Cut gadgets for the other generalized RMB checks (checking if $b_a \neq c$, $b_a = c + a$, or $b_a \neq c+a$) can be defined similarly. The following lemma (similar to Lemma 4.7) shows how to use the above forms of gadgets to derive a reduction from NP to Gap-MaxCUT.

LEMMA 4.15 (MaxCUT implementation of a verifier). *Let V be a verifier for L of logarithmic randomness, with perfect completeness and soundness s, such that V performs either a single parity check (with probability q) or a single RMB check (with probability $1-q$). Here, we refer to the generalized checks as defined in Proposition* 4.3. *Furthermore suppose that, in either case, the verifier never makes two identical queries. If there exists an $(\alpha_1 - \beta, \beta)$-PC gadget consisting of edges of total weight $w_1 - \beta$ and an $(\alpha_2 - \beta, \beta)$-RMBC gadget consisting of edges of total weight $w_2 - \beta$, then L reduces to* Gap-MaxCUT$_{c',s'}$ *for $c' = \frac{\alpha_1 q + \alpha_2(1-q)}{w_1 q + w_2(1-q)}$ and $s' = \frac{\alpha_1 q + \alpha_2(1-q) - (1-s)\beta}{w_1 q + w_2(1-q)}$. In particular, $c'/s' \geq 1 + \frac{(1-s)\beta}{\alpha_1 q + \alpha_2(1-q) - (1-s)\beta}$.*

Remark 4.16. Actually, the conclusion of the lemma holds provided all the *generalized* parity check (resp., RMB check) functions have (α_1, β) gadgets (resp., (α_2, β) gadgets).

Proof. Let PC-CUT$(a, b, c, T, x_1, \ldots, x_{n_1})$ denote the parity check gadget and RMBC-CUT$(a, b_0, b_1, c, T, x_1, \ldots, x_{n_2})$ denote the RMBC gadget. These are simplified gadgets as defined above. Increasing the α value by β, we can easily obtain the general gadgets as defined in Proposition 4.3. For example, to check that $a+b+c=1$, we introduce a gadget which, in addition to the variables $a, b, c, T, x_1, \ldots, x_{n_1}$, has an auxiliary vertex denoted $\bar{a}$. The new gadget consists of the edge $(a, \bar{a})$ having weight β together with the weighted graph PC-CUT$(\bar{a}, b, c, T, x_1, \ldots, x_{n_1})$. Clearly, the result is an (α_1, β) gadget of weight w_1 for $a+b+c=1$. Likewise we can check the condition $b_a + c = \tau_a$, where τ_0, τ_1 are any fixed bits as follows. In case $\tau_0 = \tau_1 = 1$, we introduce an auxiliary vertex $\bar{c}$, connect it to c by an edge of weight β, and use the graph RMBC-CUT $(a, b_0, b_1, \bar{c}, T, x_1, \ldots, x_{n_2})$. In case $\tau_0 = 0$ and $\tau_1 = 1$, we introduce an auxiliary vertex $\bar{b}_1$, connect it to b_1 by an edge of weight β, and use the graph RMBC-CUT$(a, b_0, \bar{b}_1, c, T, x_1, \ldots, x_{n_2})$. The case $\tau_0 = 1$ and $\tau_1 = 0$ is analogous, whereas $\tau_0 = \tau_1 = 0$ is obtained by the simplified gadget itself. Thus, we have (α_2, β) gadgets for all cases of the RMB check. Throughout the rest of the proof, PC-CUT and RMBC-CUT denote the generalized gadgets.

We create a graph G_x and weight function w_x which encodes the actions of the verifier V on input x. The vertices of G_x are as follows:

(1) For every bit $\pi[q]$ of the proof queried by the verifier V, the graph G_x has a vertex $v_{\pi[q]}$.

(2) For every random string R tossed by the verifier V, we create vertices $v_{R,i}$, for i going from 1 to $\max\{n_1, n_2\}$.

(3) There will be one special vertex T.

The edges of G_x are defined by the various gadgets. We stress that the same edge may appear in different gadgets (and its weight in these gadgets may be different). The graph G_x is defined by taking all these edges and thus it is a graph (or multigraph) with parallel edges and weights. The natural conversion of G_x into a graph with no parallel edges replaces the parallel edges between two vertices with a single

edge whose weight is the sum of the weights of the original edges. Alternatively, since the weights are constants which do not depend on x, we can transform G_x into a unweighted graph with parallel edges.

Suppose that on random string R the verifier V queries the oracle for bits $\pi[q_1]$, $\pi[q_2]$, and $\pi[q_3]$ and then does a parity check on these three bits. Then, corresponding to this random string, we add the weighted edges of the graph G_R to the graph G_x where $G_R = \text{PC-CUT}(v_{\pi[q_1]}, v_{\pi[q_2]}, v_{\pi[q_3]}, T; v_{R,1}, \ldots, v_{R,n_1})$. Alternatively, if the verifier V performs a respect of monomial basis test on the bits $\pi[q_1]$, $\pi[q_2]$, $\pi[q_3]$, and $\pi[q_4]$, then we add the weighted edges of the graph $G_R = \text{RMBC-CUT}(v_{\pi[q_1]}, v_{\pi[q_2]}, v_{\pi[q_3]}, v_{\pi[q_4]}, T; v_{R,1}, \ldots, v_{R,n_2})$.

Let N denote the number of possible random strings used by V. Observe that the total weight of the edges of G_x is $w_1 qN + w_2(1-q)N$. We now analyze the value of $\text{MaxCUT}(G_x)$.

If $x \in L$, then there exists an oracle π such that $V^\pi(x)$ always accepts. We define a cut $(S, \bar{S})$ in G_x in the following way: we place $T \in S$ and for every query q we place $v_{\pi[q]} \in S$ iff $\pi[q] = 1$. Then for each R, there exists a placement of the vertices $v_{R,i}$ so that the size of the cut induced in G_R is α_1 if R corresponds to V performing a parity check and α_2 if R corresponds to V performing an RMB check. The weight of the so obtained cut is $\alpha_1 qN + \alpha_2(1-q)N$.

Now consider $x \notin L$. We claim that if there exists a cut $(S, \bar{S})$ such that the weight of the cut is greater than $qN\alpha_1 + (1-q)N\alpha_2 - (1-s)N\beta$, then there exists an oracle π, such that $V^\pi(x)$ accepts with probability at least s. Since we know this cannot happen, we conclude that $\text{MaxCUT}(G_x) < qN\alpha_1 + (1-q)N\alpha_2 - (1-s)N\beta$. To prove the claim, we convert any cut in G_x into an oracle π where $\pi[q] = 1$ iff T and $v_{\pi[q]}$ lie on the same side of the cut. Now, by the property of the gadgets, if a graph $G_R = \text{PC-CUT}(y[q_1], y[q_2], y[q_3], T; x_1, \ldots, x_{n_1})$ contributes more than a weight of $\alpha_1 - \beta$ to the cut, then V accepts π on random string R. (Similarly, if the graph G_R is an RMBC gadget and contributes more than $\alpha_2 - \beta$ to the cut, then V accepts π on random string R.) Recall that no gadget can contribute more than the corresponding α to any cut. Thus if the total weight of the cut is more than $(\alpha_1 - \beta)qN + (\alpha_2 - \beta)(1-q)N + sN \cdot \beta$, then V accepts on at least sN random strings. This proves the claim and the lemma follows. □

We now turn to the construction of cut gadgets. Our first gadget, denoted

$$\text{PC-CUT}(a, b, c, T; \textsc{Aux}) ,$$

is a complete graph defined on five vertices $\{a, b, c, T, \textsc{Aux}\}$. The weight function, w, assigns to the edge $\{u, v\}$ weight $w_u \cdot w_v$, where $w_a = w_b = w_c = w_T = 1$ and $w_{\textsc{Aux}} = 2$. The following claim shows how $\text{PC-CUT}(a, b, c, T; \textsc{Aux})$ functions as a parity check gadget.

CLAIM 4.17 (MaxCUT PC gadget). *PC-CUT*$(a, b, c, T; \textsc{Aux})$ *is a* $(9, 1)$*-parity check gadget consisting of edges of total weight* 14.

The straightforward (but tedious) proof is omitted (and can be found in [20]).

The second gadget, denoted $\text{RMBC-CUT}(a, b_0, b_1, c, T; \textsc{Aux}_1, \textsc{Aux}_2, \textsc{Aux}_3, a')$, is composed of two graphs denoted G_1 and G_2, respectively. To motivate the construction we first observe that the condition $b_a = c$ (i.e., $(a = 0) \Rightarrow (b_0 = c)$ and $(a = 1) \Rightarrow (b_1 = c)$) is equivalent to the conjunction of $(b_0 = b_1) \Rightarrow (b_0 = c)$ and $(b_0 \neq b_1) \Rightarrow (a + b_0 + c = 0)$. The graph $G_1(b_0, b_1, c; \textsc{Aux}_1)$ will take care of the first implication. It consists of the vertex set $\{b_0, b_1, c, \textsc{Aux}_1\}$, the unit-weight edges $\{b_0, \textsc{Aux}_1\}$ and $\{b_1, \textsc{Aux}_1\}$, and a weight 2 edge $\{c, \textsc{Aux}_1\}$. The graph

$G_2(a, b_0, b_1, c, T; \text{AUX}_2, \text{AUX}_3, a')$, taking care of the second implication, consists of two subgraphs PC-CUT$(a, b_0, c, T; \text{AUX}_2)$ and $\overline{\text{PC-CUT}}(a, b_1, c, T; \text{AUX}_3, a')$, where the latter is supposed to "check" $a + b_1 + c = 1$. Specifically, $\overline{\text{PC-CUT}}(a, b, c, T; \text{AUX}, a')$ consists of the graph PC-CUT$(a', b, c, T; \text{AUX})$ and a unit-weight edge $\{a, a'\}$. The following claim shows exactly how good this gadget is in "verifying" that $b_a = c$.

CLAIM 4.18 (MaxCUT RMB gadget). *RMBC-CUT*$(a, b_0, b_1, c, T; \text{AUX}_1, \text{AUX}_2, \text{AUX}_3, a')$ *is a* $(22, 2)$*-RMBC gadget consisting of edges of total weight* 33.

Again, the straightforward proof is omitted (and can be found in [20]).

Proof of Theorem 4.14. The theorem follows by combining Proposition 4.3, Lemma 4.15, and Claims 4.17 and 4.18 (when regarding the RMB gadget as a $(11, 1)$ gadget rather than a $(22, 2)$ gadget). Details follows.

As in the proof of Theorem 4.6, when applying Lemma 4.15 to the verifier in Proposition 4.3, we obtain the same expression for the gap, c'/s', for which NP $\leq_D^\kappa$ Gap-MaxCUT$_{c',s'}$; namely,

$$\begin{aligned}\frac{c'}{s'} &\to 1 + \frac{(1-s)\beta}{q \cdot \alpha_1 + (1-q) \cdot \alpha_2 - (1-s)\beta} \\ &= 1 + \frac{3}{12\alpha_1 + 8\alpha_2 - 3}.\end{aligned}$$

Recall that here $\alpha_1 - 1 = 9$ and $\alpha_2 - 1 = 11$ (rather than $\alpha_1 = 9$ and $\alpha_2 = 11$; see Lemma 4.15). The above simplifies to $1 + \frac{3}{213} = 72/71$ and the bound on $\frac{c'}{s'}$ follows. As for c', it equals $\frac{3\alpha_1 + 2\alpha_2}{3w_1 + 2w_2} > 0.6$ (as $w_1 - 1 = 14$ and $w_2 - 1 = 16.5$). □

5. Free bits and vertex cover. It is known that approximating the minimum vertex cover of a graph to within a $1+\epsilon$ factor is hard, for some $\epsilon > 0$ [76, 8]. However, we do not know of any previous attempt to provide a lower bound for ϵ. An initial attempt may use VC gadgets that implement the various tests in V_{SNPinner} analogously to the way it was done in the previous sections for the MaxSAT versions and MaxCUT. This yields a lower bound of $\epsilon > \frac{1}{43} > 0.023$ (see details in previous versions of this work [20]). However, a stronger result is obtained via free-bit complexity:[9] We apply the FGLSS reduction to a proof system (for NP) of low free-bit complexity, specifically to a proof system which uses two free bits and has soundness error below 0.8. Consequently, the clique size, in case the original input is in the language, is at least one-fourth (1/4) of the size of the graph, which means that translating clique-approximation factors to VC-approximation factors yields only a loss of a factor of 3. Since the FGLSS transformation translates the completeness/soundness ratio to the gap factor for approximating clique, our first goal is to construct for NP a proof system which uses two free bits and has soundness error as low as possible. We remark that the proof system of section 6 uses seven free bits and achieves soundness error less than 1/2. The reader may observe that, following the above approach, it is not worthwhile to use the proof system of section 6 or any proof systems which achieves a soundness error of 1/2 at the cost of five free bits or more. On the other hand, in light of the results of section 10, we cannot hope for a proof system of free-bit complexity 1 (and perfect completeness) for NP.

5.1. Minimizing the error achievable with two free bits. The pcp system of Proposition 4.3 had free-bit complexity 2 (and query complexity 3). However, a

[9] Furthermore, there seems to be little hope that the former approach can ever yield an improvement over the better bounds subsequently obtained by Håstad [57].

The enhanced RMB test. Again, $A\colon \mathcal{F}_l \to \Sigma$ is the object being tested, and the test take additional inputs or parameters $f_1, f_2 \in \mathcal{F}_l$.

EMBTest$(A; f_1, f_2)$ (enhanced monomial-basis test).
For every $f \in \mathcal{F}_l$, invoke **MBTest**$(A; f_1, f, f_2)$.
Output 0 if all invocations answered with 0, else output 1.

The passing probability:

$$\textsc{embPass}(A) = \Pr_{f_1, f_2 \stackrel{R}{\leftarrow} \mathcal{F}_l}[\,\mathbf{EMBTest}(A; f_1, f_2) = 0\,].$$

FIG. 12. *The enhanced RMB test and its passing probability.*

The two free-bit inner verifier. Given functions $h \in \mathcal{F}_l$ and $\sigma\colon \Sigma^l \to \Sigma^{l_1}$, the verifier has access to oracles for $A\colon \mathcal{F}_l \to \Sigma$ and $A_1\colon \mathcal{F}_{l_1} \to \Sigma$. In addition it takes a parameter $p \in [0,1]$.

Pick $q \stackrel{R}{\leftarrow} [0,1]$.

Case: $q \le p$:
Pick $f_1, f_2 \stackrel{R}{\leftarrow} \mathcal{F}_l$.
LinTest$(A_{(h,0),(\bar{1},1)}; f_1, f_2)$.
EMBTest$(A_{(h,0),(\bar{1},1)}; f_1, f_2)$.

Case: $q > p$:
Pick $f \stackrel{R}{\leftarrow} \mathcal{F}_l$ and $g \stackrel{R}{\leftarrow} \mathcal{F}_{l_1}$.
ProjTest$_\sigma(A_{(h,0),(\bar{1},1)}, A_1; f, g)$.

Remark: access to $A_{(h,0),(\bar{1},1)}(f)$ is implemented by accessing either $A(f)$ or $A(f+h)$ or $A(f+\bar{1})$ or $A(f+h+\bar{1})$.

FIG. 13. *The two free-bit inner verifier* $V_{2\text{inner}}$.

smaller soundness error can be achieved if we make more queries. Our starting point is part 2 of Lemma 3.19 which suggests an RMB test with a detection probability that is twice as big, still using 2 free bits (alas, $2^l + 2$ rather than three queries). Specifically, we consider an *enhanced* RMB test which, on input $f_1, f_2 \in \mathcal{F}_l$, goes over all $f \in \mathcal{F}_l$ invoking the atomic RMB test with input functions f_1, f, f_2. The enhanced RMB Test, denoted **EMBTest**, is depicted in Figure 12. Further improvement is obtained by "packing" together the linearity test and the enhanced RMB test (in contrast to V_{SNPinner} in which these tests were performed exclusively). Both tests make three queries of which two are common, and the answers to these queries determine the answer to the third query (which is different in the two tests). The resulting inner verifier, denoted $V_{2\text{inner}}$, is depicted in Figure 13. As V_{SNPinner}, the verifier $V_{2\text{inner}}$ works with functions/oracles A that are folded twice—once across $(h,0)$ and once across $(\bar{1},1)$.

The following corollary is immediate from part 2 of Lemma 3.19.

COROLLARY 5.1 (analysis of the enhanced monomial-basis test). *Let* $A, \tilde{A}\colon \mathcal{F}_l \to \Sigma$ *with* A *satisfying* $A(f+\bar{1}) = A(f)+1$ *for all* f *and* $\tilde{A}$ *linear but not respecting the*

monomial basis. Let $x = \mathrm{Dist}(A, \tilde{A})$. *Then*

$$1 - \textsc{embPass}(A) \;\geq\; \frac{3}{4} \cdot (1 - 2x).$$

The following lemma is analogous to Lemma 4.1. Loosely speaking, it considers three possible strategies of a "dishonest" prover and indicates the probability with which the verifier detects an error.

LEMMA 5.2 (soundness of $V_{2\mathrm{inner}}$). *Let* $\delta_1, \delta_2 > 0$, $0 \leq p \leq 1$, *and* $l, l_1 \in \mathcal{Z}^+$. *Then the* (l, l_1)*-canonical inner verifier* $V_{2\mathrm{inner}}$ *(with parameter* p*) is* $(\rho, \delta_1, \delta_2)$*-good, where* $1 - \rho = \min(T_1, T_2, T_3)$ *and*

(1) $T_1 \stackrel{\mathrm{def}}{=} (1/2 - \delta_1) \cdot p$,

(2) $T_2 \stackrel{\mathrm{def}}{=} p \cdot \min_{x \leq 1/2 - \delta_1} [\, \max(\Gamma_{\mathrm{lin}}(x)\,,\, 3/4 \cdot (1 - 2x)) \,]$,

(3) $T_3 \stackrel{\mathrm{def}}{=} \min_{x \leq 1/2 - \delta_1} [\, p \cdot \Gamma_{\mathrm{lin}}(x) + (1 - p) \cdot (1/2 - \delta_2)(1 - 2x) \,]$.

Proof. The analysis is broken up into several cases as in the proof of Lemma 4.1. Let $x = \mathrm{Dist}(A_{(h,0),(\bar{1},1)}, \textsc{Lin})$.

Case 1. $x \geq 1/2 - \delta_1$. Lemma 3.15 implies that $1 - \textsc{LinPass}(A_{(h,0),(\bar{1},1)}) \geq \Gamma_{\mathrm{lin}}(x) \geq x \geq 1/2 - \delta_1$. Since $V_{2\mathrm{inner}}$ performs the atomic linearity test with probability p, we have in this case

$$1 - \mathtt{ACC}\,[\, V_{2\mathrm{inner}}^{A,A_1}(\sigma, h)\,] \geq p \cdot (1/2 - \delta_1).$$

Case 2. $x < 1/2 - \delta_1$. Again, Lemma 3.15 implies that $1 - \textsc{LinPass}(A_{(h,0),(\bar{1},1)}) \geq \Gamma_{\mathrm{lin}}(x)$, and

$$1 - \mathtt{ACC}\,[\, V_{2\mathrm{inner}}^{A,A_1}(\sigma, h)\,] \;\geq\; p \cdot \Gamma_{\mathrm{lin}}(x)$$

follows. Now let $\tilde{A}$ be a linear function such that $\mathrm{Dist}(A_{(h,0),(\bar{1},1)}, \tilde{A}) = x$. We consider the following subcases.

Case 2.1. $\tilde{A}$ does not respect the monomial basis. In this case Corollary 5.1 implies that $1 - \textsc{embPass}(A_{(h,0),(\bar{1},1)}) \geq \frac{3}{4} \cdot (1 - 2x)$. So the probability that $V_{2\mathrm{inner}}$ rejects is at least $p \cdot \frac{3}{4} \cdot (1 - 2x)$. Combining the two lower bounds on $1 - \mathtt{ACC}\,[\, V_{2\mathrm{inner}}^{A,A_1}(\sigma, h)\,]$, we get

$$1 - \mathtt{ACC}\,[\, V_{2\mathrm{inner}}^{A,A_1}(\sigma, h)\,] \;\geq\; p \cdot \max\left(\Gamma_{\mathrm{lin}}(x), \frac{3}{4}(1 - 2x)\right).$$

Case 2.2. $\tilde{A}$ respects the monomial basis. By Proposition 3.2, $\tilde{A}$ is an evaluation operator. So there exists $a \in \Sigma^l$ such that $\tilde{A} = E_a$. So $\mathrm{Dist}(A_{(h,0),(\bar{1},1)}, E_a) = x$. Let $a_1 = \sigma(a)$. The proof splits into two further subcases.

Case 2.2.1. $d \stackrel{\mathrm{def}}{=} \mathrm{Dist}(A_1, E_{a_1}) \geq 1/2 - \delta_2$. By Lemma 3.21 we have $1 - \textsc{ProjPass}_{\sigma}(A_{(h,0),(\bar{1},1)}, A_1) \geq d \cdot (1 - 2x) \geq (1/2 - \delta_2) \cdot (1 - 2x)$. So the probability that $V_{2\mathrm{inner}}$ performs the projection test and rejects is at least $(1 - p) \cdot (1/2 - \delta_2)(1 - 2x)$. To this we add the probability of the exclusively disjoint event in which the verifier performs the linearity test and rejects, obtaining

$$1 - \mathtt{ACC}\,[\, V_{2\mathrm{inner}}^{A,A_1}(\sigma, h)\,] \;\geq\; p \cdot \Gamma_{\mathrm{lin}}(x) + (1 - p) \cdot (1/2 - \delta_2)(1 - 2x).$$

Case 2.2.2. In this case, we have $x = \mathrm{Dist}(A_{(h,0),(\bar{1},1)}, E_a) < 1/2 - \delta_1$ and $\mathrm{Dist}(A_1, E_{a_1}) < 1/2 - \delta_2$. Thus the functions $A_{(h,0),(\bar{1},1)}$ and A_1 satisfy conditions (2.1) and (2.2) in Definition 3.9.

Similarly to the proof of Lemma 4.1, we infer that the lower bound on $1-\rho$ is as claimed and the lemma follows. □

We now simplify the soundness bound of the lemma. The proof of the first item uses the fact that $\Gamma_{\mathrm{lin}}(x) \geq 45/128$ for all $x \geq \frac{1}{4}$. The second item uses the fact that $\Gamma_{\mathrm{lin}}(x) \geq x$ for all $x \leq 1/2$.

CLAIM 5.3.

(1) $\min_{x \leq 1/2-\delta_1} [\max(\Gamma_{\mathrm{lin}}(x)\,,\, \frac{3}{4}(1-2x))] \geq 45/128$.

(2) $\min_{x \leq 1/2-\delta_1} [\, p \cdot \Gamma_{\mathrm{lin}}(x) + (1-p) \cdot (1/2 - x)\,] \geq \frac{1}{2} \cdot \min(p\,,\, 1-p)$.

(3) *Let T_1, T_2 and T_3 be as in Lemma* 5.2. *Then*

$$\min(T_1, T_2, T_3) \;\geq\; \min\left\{ \frac{45}{128} \cdot p\,,\; \frac{1}{2} \cdot (1-p) \right\} - \max(\delta_1, \delta_2).$$

Interestingly, the lower bound provided by item 3 is tight. Optimization calls for setting $\frac{45}{128} \cdot p = \frac{1}{2} \cdot (1-p)$, which yields $p = \frac{64}{109}$ and a soundness bound of $1 - \frac{45}{128}p + \max(\delta_1, \delta_2) = 1 - \frac{45}{218} + \max(\delta_1, \delta_2)$.

Proof. Toward proving part 1 we consider two cases.

Case 1.1. $x \geq 1/4$. In this case, by definition of Γ_{lin}, we have

$$\max(\Gamma_{\mathrm{lin}}(x)\,,\, \frac{3}{4}(1-2x)) \;\geq\; \Gamma_{\mathrm{lin}}(x) \;\geq\; \frac{45}{128}.$$

Case 1.2. $x \leq 1/4$. In this case we have

$$\max\left(\Gamma_{\mathrm{lin}}(x)\,,\, \frac{3}{4}(1-2x)\right) \;\geq\; \frac{3}{4}(1-2x) \;\geq\; \frac{3}{8} \;>\; \frac{45}{128}.$$

This establishes part 1. Towards proving part 2 we consider two different cases.

Case 2.1. $p \leq (1-p)$. In this case,

$$p \cdot \Gamma_{\mathrm{lin}}(x) + (1-p) \cdot \left(\frac{1}{2} - x\right) \;\geq\; p \cdot x + p \cdot \left(\frac{1}{2} - x\right) \;=\; \frac{p}{2}.$$

Case 2.2. $p \geq (1-p)$. In this case,

$$p \cdot \Gamma_{\mathrm{lin}}(x) + (1-p) \cdot \left(\frac{1}{2} - x\right) \;\geq\; (1-p) \cdot x + (1-p) \cdot \left(\frac{1}{2} - x\right) \;=\; \frac{1-p}{2}.$$

This establishes part 2. To prove part 3 use parts 1 and 2 to lower bound T_2 and T_3, respectively, and get

$$\begin{aligned} \min(T_1, T_2, T_3) &\geq \min\left\{ \left(\frac{1}{2} - \delta_1\right) \cdot p\,,\; \frac{45}{128} \cdot p\,,\; \frac{1}{2} \cdot \min(p\,,\, 1-p) - \delta_2 \right\} \\ &\geq \min\left\{ \frac{45}{128} \cdot p\,,\; \frac{1}{2} \cdot (1-p) \right\} - \max(\delta_1, \delta_2). \end{aligned}$$

The claim follows. □

Composing the above inner verifier with an adequate outer verifier, we get the following.

THEOREM 5.4. *For any* $s > \frac{173}{218} \approx 0.79357798$, $\mathrm{NP} \subseteq \mathrm{FPCP}_{1,s}[\log, 2]$, *and furthermore, there is a constant* q *such that* $\mathrm{NP} \subseteq \mathrm{PCP}_{1,s}[\,\mathsf{coins} = \log\,;\,\mathsf{free} = 2\,;\,\mathsf{query} = q\,]$.

Proof. Let $\delta = s - \frac{173}{218}$, $\delta_1 = \delta_2 = \delta/3$, and $\epsilon = \delta/3 \cdot 16\delta_1^2\delta_2^2 = \frac{16\delta^5}{243}$. Now, let l and l_1 be integers such that the outer verifier, V_{outer}, guaranteed by Lemma 3.8, is (l, l_1)-canonical and ϵ-good for $L \in \mathrm{NP}$. Consider the (l, l_1)-canonical inner verifier $V_{2\mathrm{inner}}$ working with parameter $p = \frac{64}{109}$. Using Lemma 5.2 and Claim 5.3, we conclude that $V_{2\mathrm{inner}}$ is (ρ, δ, δ)-good for $\rho = 1 - 45/218 + \max(\delta_1, \delta_2)$.

Composing V_{outer} and $V_{2\mathrm{inner}}$ we obtain a verifier, $V_{2\mathrm{free}}$, which by Theorem 3.12 has soundness error bounded above by $\frac{173}{218} + \max(\delta_1, \delta_2) + \frac{\epsilon}{16\delta_1^2\delta_2^2} = s$, as required. Furthermore, $V_{2\mathrm{free}}$ uses logarithmically many coins. We claim that $V_{2\mathrm{free}}$ has query complexity $2^{2^l} + 2$ and free-bit complexity 2. The claim is obvious in case $V_{2\mathrm{inner}}$ performs the projection test. Otherwise, $V_{2\mathrm{inner}}$ performs a linearity test with parameters f_1 and f_2 and an enhanced RMB test with the same parameters. Clearly, the answers on f_1 and f_2 determine the acceptable (by linearity test) answer on $f_1 + f_2$. The key observation is that the former two answers also determine all 2^{2^l} acceptable answers in the enhanced RMB test (i.e., for every $f \in \mathcal{F}_l$, the answer on $f_1' \cdot f + f_2$ should equal the answer on f_2, where $f_1' = f_1$ if the answer on f_1 is zero and $f_1' = f_1 + \bar{1}$ otherwise). □

By repeating the above proof system three times, we obtain the following.

COROLLARY 5.5. $\mathrm{NP} \subseteq \mathrm{FPCP}_{1,1/2}[\log, 6]$. *Furthermore, there is a constant* q *such that*

$$\mathrm{NP} \subseteq \mathrm{PCP}_{1,1/2}[\,\mathsf{coins} = \log\,;\,\mathsf{free} = 6\,;\,\mathsf{query} = q\,]\,.$$

Proof. There exists $\epsilon > 0$ such that $\left(\frac{173}{218} + \epsilon\right)^3 \leq 1/2$. □

5.2. Hardness of vertex cover. Refer to section 2.4 for the definition of the MinVC problem and the associated gap problem $\mathsf{Gap}\text{-}\mathrm{MinVC}_{c,s}$, and to section 2.4.3 for status and previous work.

GOING FROM FREE BITS TO VC. Instead of reducing from Max3SAT, we first use Theorem 5.4 to get gaps in clique size and then apply the standard reduction.

PROPOSITION 5.6. $\mathrm{FPCP}_{c,s}[\log, f] \leq_D^K \mathsf{Gap}\text{-}\mathrm{MinVC}_{c',s'}$ *for* $s' = 1 - 2^{-f}c$ *and* $\frac{c'}{s'} = 1 + \frac{c-s}{2^f - c}$.

Proof. The FGLSS reduction says that $\mathrm{FPCP}_{c,s}[\log, f] \leq_D^K \mathsf{Gap}\text{-}\mathrm{MaxClique}_{c'',s''}$, where $c'' = 2^{-f} \cdot c$ and $s'' = 2^{-f} \cdot s$. (See section 2.4 for definition of Gap-MaxClique.) Now we apply the standard Karp reduction (of MaxClique to MinVC) which maps a graph G to its complement $\overline{G}$, noting that $\overline{\mathrm{MinVC}}(\overline{G}) = 1 - \overline{\mathrm{MaxClique}}(G)$. Thus $\mathsf{Gap}\text{-}\mathrm{MaxClique}_{c'',s''} \leq_D^K \mathsf{Gap}\text{-}\mathrm{MinVC}_{1-s'',1-c''}$. Now set $c' = 1 - s''$ and $s' = 1 - c''$, and note

$$\frac{c'}{s'} \;=\; \frac{1 - s''}{1 - c''} \;=\; \frac{1 - s2^{-f}}{1 - c2^{-f}} \;=\; 1 + \frac{c - s}{2^f - c}\,.$$

This completes the proof. □

OUR RESULTS. We obtain the first explicit and reasonable constant factor non-approximability result for MinVC. A consequence of the following theorem is that, assuming P ≠ NP, there is no polynomial-time algorithm to approximate MinVC within a factor of 1.0688.

THEOREM 5.7. $\mathsf{Gap}\text{-}\mathrm{MinVC}_{c,s}$ *is* NP*-complete for some* c, s *satisfying* $c/s \geq 1.0688 > 16/15$. *Moreover,* $s = 3/4$.

Proof. The proof follows immediately from Proposition 5.6 and Theorem 5.4. Namely, for any $s' > \frac{173}{218}$, $\mathrm{NP} \subseteq \mathrm{FPCP}_{1,s'}[\log, 2] \leq_D^K \mathsf{Gap}\text{-}\mathrm{MinVC}_{c,s}$ for $s = 1-2^{-2} = 3/4$ and $\frac{c}{s} = 1 + \frac{1-s'}{2^2-1} = 1 + \frac{1-s'}{3}$. Thus, $\frac{c}{s} = 1 + \frac{15}{218} - \frac{\epsilon}{3} > 1.068807 - \frac{\epsilon}{3}$, where $\epsilon \stackrel{\mathrm{def}}{=} s' - \frac{173}{218}$. □

We remark that a special case of Proposition 5.6, in which the statement is restricted to $f = 0$, would have sufficed for proving the above theorem, the reason being that we could have applied Proposition 11.8 to Theorem 5.4 and obtained $\mathrm{NP} \subseteq \mathrm{FPCP}_{1/4,s/4}[\log, 0]$, for $s = 0.7936$, which by the special case of Proposition 5.6 is reducible to $\mathsf{Gap}\text{-}\mathrm{MinVC}_{c',s'}$ with $s' = 1-1/4 = 3/4$ and $\frac{c'}{s'} = 1 + \frac{(1/4)-(s/4)}{1-(1/4)} = 1 + \frac{1-s}{3}$ (as above). Interestingly, the special case of Proposition 5.6 can be "reversed," namely, $\mathsf{Gap}\text{-}\mathrm{MinVC}_{c',s'}$ is reducible to $\mathrm{FPCP}_{c,s}[\log, 0]$ with $s = 1-c'$, $c = 1-s'$, and $\frac{c}{s} = \frac{1-s'}{1-c'}$ (which reverses $\frac{c'}{s'} = \frac{1-s}{1-c} = 1 + \frac{c-s}{1-c}$). The key fact in proving this "reverse reduction" is Corollary 8.5 which asserts that $\mathsf{Gap}\text{-}\mathrm{MaxClique}_{c,s} \leq_D^K \mathrm{FPCP}_{c,s}[\log, 0]$. However, we do not know if it is possible to "reverse" the other step in the alternative proof, namely, whether $\mathrm{FPCP}_{c,s}[\log, 0]$ is reducible to $\mathrm{FPCP}_{4c,4s}[\log, 2]$ (our reverse transformation is weaker; see Proposition 11.6).

6. Minimizing the number of queries for soundness 0.5. The problem we consider here is to minimize the values of q (and q_{av}) for which we can construct PCPs for NP using q queries in the worst case (and q_{av} on the average) to achieve a soundness error of $1/2$. We allow only logarithmic randomness. See section 2.2.3 for description of past records.

Sources of our improvements. The principal part of our improvement comes from the use of the new long code based inner verifier, the atomic tests and their analysis in section 3.5, and the new idea of folding. By repeating the proof system of Theorem 4.5 five times, we obtain that Eq. (4) holds for $q = 15$. (Note that $5 = \min\{i \in \mathsf{N} : 0.85^i < 0.5\}$.) A straightforward implementation of the recycling technique of [21] yields that Eq. (4) holds for $q = 12$ and $q_{\mathrm{av}} = 11.74$. Using a more careful implementation of this technique, we reduce the query complexity by an additional bit.

6.1. The PCP inner verifier. Our result is based on the construction of the (l, l_1)-canonical inner verifier V_{PCPinner} depicted in Figure 14. In addition to its standard inputs h, σ, it takes parameters $p_1, p_2, p_3 \geq 0$ so that $p_1 + p_2 + p_3 = 1$. The inner verifier V_{PCPinner} combines the atomic tests in three different ways.

(1) Some tests are performed independently (i.e., the main steps in Figure 14);
(2) Some tests are performed while reusing some queries (i.e., the tests in Step 2 reuse f_3);
(3) Some tests are performed in a mutually exclusive manner (i.e., the tests in Step 3).

As in previous sections, the tests are executed on the function $A_{(h,0),(\bar{1},1)}$ to which the verifier has an effective oracle access given its access to A. By inspection it is clear that the total number of accesses to the oracles for A and A_1 is $3 + 5 + 3 = 11$ (whereas the free-bit complexity is $2 + 3 + 2 = 7$). We now examine the goodness of V_{PCPinner}. Recall the definitions of the functions $\Gamma_{\mathrm{lin}}(x)$ (from Lemma 3.15) and $\Gamma_{\mathrm{RMB}}(x) = \frac{3}{8}(1-2x)$ (from Lemma 3.19).

Lemma 6.1 (soundness of V_{PCPinner}). *For any $0 < \delta_1, \delta_2 < 0.1$ and any l, l_1, p_1, p_2, and p_3, satisfying $p_1 + p_2 + p_3 = 1$ and $5p_1 = 2p_2$, the (l, l_1)-canonical inner*

The PCP inner verifier. This (l, l_1)-canonical inner verifier is given functions $h \in \mathcal{F}_l$ and $\sigma\colon \Sigma^l \to \Sigma^{l_1}$, and has access to oracles for $A\colon \mathcal{F}_l \to \Sigma$ and $A_1\colon \mathcal{F}_{l_1} \to \Sigma$. In addition it takes three nonnegative parameters p_1, p_2, and p_3 which sum up to 1.

Pick functions $f_1, \ldots, f_8 \overset{R}{\leftarrow} \mathcal{F}_l$ and $g_1, g_2 \overset{R}{\leftarrow} \mathcal{F}_{l_1}$.

Step 1: Linearity test.
LinTest$(A_{(h,0),(\bar{1},1)}; f_1, f_2)$.

Step 2: Combined RMB and projection test.
MBTest$(A_{(h,0),(\bar{1},1)}; f_3, f_4, f_5)$.
ProjTest$_\sigma(A_{(h,0),(\bar{1},1)}, A_1; f_3, g_1)$.

Step 3: Invoking V_{SNPinner} with parameters p_1, p_2, p_3.
Pick $p \overset{R}{\leftarrow} [0,1]$.

Case $p \le p_1$:	**LinTest**$(A_{(h,0),(\bar{1},1)}; f_6, f_7)$.
Case $p_1 < p \le p_1 + p_2$:	**MBTest**$(A_{(h,0),(\bar{1},1)}; f_6, f_7, f_8)$.
Case $p_1 + p_2 < p$:	**ProjTest**$_\sigma(A_{(h,0),(\bar{1},1)}, A_1; f_6, g_2)$.

Accept iff all the above tests accept.

Remark: access to $A_{(h,0),(\bar{1},1)}(f)$ is implemented by accessing either $A(f)$, $A(f+h)$, $A(f+\bar{1})$, or $A(f+h+\bar{1})$.

FIG. 14. *The PCP inner verifier* V_{PCPinner}.

verifier V_{PCPinner} *is* $(\rho, \delta_1, \delta_2)$*-good, where* $1-\rho$ *is the minimum of the following three quantities:*

(1) $1/2 + \frac{p_1}{10} - \delta_1$;
(2) $1 - (11/14)^3 - \frac{p_3}{1-p_3} > 0.51494168 - \frac{p_3}{1-p_3}$;
(3) $\min\{1/2 + \frac{p_3}{20} - \delta_2\,,\, 1 - (0.55218507 + \delta_2) \cdot (1 - 45/128 p_1)\}$.

Furthermore, if $p_1 > 10\delta_1$, $p_3 > 20\delta_2$ *and* $p_3 \le 0.01$, *then* $1 - \rho > 1/2$.

Proof. We split the analysis into several cases based on the value of

$$x \;=\; \text{Dist}(A_{(h,0),(\bar{1},1)}, \textsc{Lin})\,.$$

Case 1. $x \ge 1/2 - \delta_1$. Lemma 3.15 implies that $\textsc{LinPass}(A_{(h,0),(\bar{1},1)}) \le 1 - \Gamma_{\text{lin}}(x) \le 1 - x \le 1/2 + \delta_1$. Thus, in this case

$$\texttt{ACC}\,[\,V^{A,A_1}_{\text{PCPinner}}(\sigma, h)\,] \;\le\; \rho_1 \overset{\text{def}}{=} (1-p_1)\cdot\left(\frac{1}{2}+\delta_1\right) + p_1 \cdot \left(\frac{1}{2}+\delta_1\right)^2 \;<\; \frac{1}{2}+\delta_1 - \frac{p_1}{10}.$$

(The last inequality is due to $\delta_1 < 0.1$.) Using $p_1 > 10\delta_1$ we get $\rho_1 < 1/2$.

Case 2. $x < 1/2 - \delta_1$. Let $\tilde{A}\colon \mathcal{F}_l \to \Sigma$ be a linear function such that $\text{Dist}(A_{(h,0),(\bar{1},1)}, \tilde{A}) = x$. The proof splits into two subcases.

Case 2.1. $\tilde{A}$ does not respect the monomial basis. In this case, by Lemmas 3.15 and 3.19 we have, using $\Gamma_{\text{lin}}(x)$, $\Gamma_{\text{RMB}}(x) < 1$, in the second inequality,

$$\begin{aligned}\mathtt{ACC}\,[\,V^{A,A_1}_{\mathrm{PCPinner}}(\sigma,h)\,] &\le (1-\Gamma_{\mathrm{lin}}(x))\cdot(1-\Gamma_{\mathrm{RMB}}(x))\cdot(1-p_1\Gamma_{\mathrm{lin}}(x)-p_2\Gamma_{\mathrm{RMB}}(x))\\ &< (1-\Gamma_{\mathrm{lin}}(x))\cdot(1-\Gamma_{\mathrm{RMB}}(x))\\ &\quad\cdot\left(1-\frac{p_1}{p_1+p_2}\cdot\Gamma_{\mathrm{lin}}(x)-\frac{p_2}{p_1+p_2}\cdot\Gamma_{\mathrm{RMB}}(x)+\frac{p_3}{1-p_3}\right)\\ &< \alpha\cdot\beta\cdot[q\alpha+(1-q)\beta]+\frac{p_3}{1-p_3},\end{aligned}$$

where $q \stackrel{\mathrm{def}}{=} \frac{p_1}{p_1+p_2}$, $\alpha = 1-\Gamma_{\mathrm{lin}}(x)$, and $\beta = 1-\Gamma_{\mathrm{RMB}}(x)$. Using $p\cdot x+(1-p)\cdot y \ge x^p\cdot y^{1-p}$, we show that $\alpha\cdot\beta\cdot[q\alpha+(1-q)\beta] \le [\frac{1+q}{3}\alpha+\frac{2-q}{3}\beta]^3$. Specifically,

$$\begin{aligned}\left[\frac{1+q}{3}\alpha+\frac{2-q}{3}\beta\right]^3 &= \left[\frac{2}{3}\cdot\left(\frac{1}{2}\alpha+\frac{1}{2}\beta\right)+\frac{1}{3}\cdot(q\alpha+(1-q)\beta)\right]^3\\ &\ge \left(\frac{1}{2}\alpha+\frac{1}{2}\beta\right)^{\frac{2}{3}\cdot 3}\cdot(q\alpha+(1-q)\beta)^{\frac{1}{3}\cdot 3}\\ &= \left[\frac{1}{2}\cdot\alpha+\frac{1}{2}\cdot\beta\right]^2\cdot(q\alpha+(1-q)\beta)\\ &\ge \alpha\cdot\beta\cdot(q\alpha+(1-q)\beta).\end{aligned}$$

Combining the above with Claim 4.2 (i.e., the lower bound on T_2), we obtain (for every $x<1/2$)

$$\begin{aligned}\mathtt{ACC}\,[\,V^{A,A_1}_{\mathrm{PCPinner}}(\sigma,h)\,] &< \left[1-\frac{1+q}{3}\cdot\Gamma_{\mathrm{lin}}(x)-\frac{2-q}{3}\cdot\Gamma_{\mathrm{RMB}}(x)\right]^3+\frac{p_3}{1-p_3}\\ &\le \left[1-\min\left(\frac{1+q}{6},\frac{2-q}{8}\right)\right]^3+\frac{p_3}{1-p_3}.\end{aligned}$$

Observe that $\min(\frac{1+q}{6},\frac{2-q}{8})$ is maximized at $q=2/7$ where its value is $3/14$. Indeed this value of q is consistent with $p_1=\frac{2}{7}\cdot(p_1+p_2)$ and so, in this case, we get

$$\mathtt{ACC}\,[\,V^{A,A_1}_{\mathrm{PCPinner}}(\sigma,h)\,] \;\le\; \rho_2 \stackrel{\mathrm{def}}{=} \left[\frac{11}{14}\right]^3+\frac{p_3}{1-p_3} \;<\; 0.48505832+\frac{p_3}{1-p_3}.$$

Using $p_3 \le 0.01$ we get $\rho_2 < 1/2$.

Case 2.2. $\tilde{A}$ respects the monomial basis. By Proposition 3.2, $\tilde{A}$ is an evaluation operator. So there exists $a\in\Sigma^l$ such that $\tilde{A}=E_a$. So $\mathrm{Dist}(A_{(h,0),(\bar{1},1)},E_a)=x$. Let $a_1=\sigma(a)$. The proof splits into two further subcases.

Case 2.2.1. $d \stackrel{\mathrm{def}}{=} \mathrm{Dist}(A_1,E_{a_1}) \ge 1/2-\delta_2$. By Lemma 3.21 we have

$$\textsc{ProjPass}_\sigma(A_{(h,0),(\bar{1},1)},A_1) \le 1-d\cdot(1-2x) < \frac{1}{2}+x+\delta_2\ .$$

Letting $\Gamma_{\mathrm{PRJ}}(x) \stackrel{\mathrm{def}}{=} 1/2-x-\delta_2$, we get in this case

$$\mathtt{ACC}\,[\,V^{A,A_1}_{\mathrm{PCPinner}}(\sigma,h)\,] \;\le\; \rho_3 \stackrel{\mathrm{def}}{=} (1-\Gamma_{\mathrm{lin}}(x))\cdot(1-\Gamma_{\mathrm{PRJ}}(x))\cdot(1-p_1\Gamma_{\mathrm{lin}}(x)-p_3\Gamma_{\mathrm{PRJ}}(x)).$$

We upper bound ρ_3 by considering three subcases (corresponding to the segments of Γ_{lin}).

Case 2.2.1.1. $x \leq 1/4$. In this case we use $\Gamma_{\text{lin}}(x) \geq 3x(1-2x)$ and obtain

$$\begin{aligned}\rho_3 &< (1-\Gamma_{\text{lin}}(x)) \cdot (1-\Gamma_{\text{PRJ}}(x)) \cdot (1-p_3\Gamma_{\text{PRJ}}(x)) \\ &< (1-3x(1-2x)) \cdot \left(\frac{1}{2}+x+\delta_2\right) \cdot \left(1-\frac{p_3}{10}\right) \\ &< \frac{1}{2} \cdot \left[1-x+12x^3\right] \cdot \left[1-\frac{p_3}{10}\right] + \delta_2 \\ &\leq \frac{1}{2} \cdot \left[1-\frac{p_3}{10}\right] + \delta_2,\end{aligned}$$

where the last inequality uses the fact that the function $x - 12x^3$ is nonnegative in the interval $[0, 1/4]$. Using $p_3 > 20\delta_2$ we obtain $\rho_3 < 1/2$.

Case 2.2.1.2. $x \geq 1/4$ and $x \leq 45/125$. In this case we use $\Gamma_{\text{lin}}(x) \geq 45/128 = \Gamma_{\text{lin}}(45/128)$ and $\Gamma_{\text{PRJ}}(x) \geq \Gamma_{\text{PRJ}}(45/128)$ and obtain

$$\begin{aligned}\rho_3 &< (1-\Gamma_{\text{lin}}(x)) \cdot (1-\Gamma_{\text{PRJ}}(x)) \cdot (1-p_1\Gamma_{\text{lin}}(x)) \\ &\leq \left(1-\Gamma_{\text{lin}}\left(\frac{45}{128}\right)\right) \cdot \left(1-\Gamma_{\text{PRJ}}\left(\frac{45}{128}\right)\right) \cdot \left(1-p_1\Gamma_{\text{lin}}\left(\frac{45}{128}\right)\right) \\ &< \frac{83}{128} \cdot \left(\frac{109}{128}+\delta_2\right) \cdot \left(1-p_1\frac{45}{128}\right) \\ &< (0.55218507+\delta_2) \cdot \left(1-p_1\frac{45}{128}\right).\end{aligned}$$

Using $\delta_2 < \frac{p_3}{10} \leq 0.001$ and $p_1 \geq \frac{2}{7} \cdot 0.99 > 0.28$, we obtain $\rho_3 < 0.5532 \cdot 0.902 < 0.499$.

Case 2.2.1.3. $x \geq 45/128$. In this case we use $\Gamma_{\text{lin}}(x) \geq x \geq 45/128$ and obtain

$$\begin{aligned}\rho_3 &< (1-\Gamma_{\text{lin}}(x)) \cdot (1-\Gamma_{\text{PRJ}}(x)) \cdot (1-p_1\Gamma_{\text{lin}}(x)) \\ &< (1-x) \cdot \left(\frac{1}{2}+x+\delta_2\right) \cdot \left(1-p_1\frac{45}{128}\right).\end{aligned}$$

The latter expression decreases in the interval $[45/128\,,\,1/2]$ and is hence maximized at $x = 45/128$. Thus we obtain the same expression as in Case 2.2.1.2, and the bound on ρ_3 follows identically.

We conclude that in Case 2.2.1 we have

$$\rho_3 \;<\; \max\left[\frac{1}{2}-\frac{p_3}{20}+\delta_2\;,\;(0.55218507+\delta_2) \cdot \left(1-p_1\frac{45}{128}\right)\right],$$

and under the hypothesis regarding p_1, p_3, and δ_2, we always have $\rho_3 < 0.5$.

Case 2.2.2. Else, we have $x = \text{Dist}(A_{(h,0),(\bar{1},1)}, E_a) \leq 1/2-\delta_1$ and $\text{Dist}(A_1, E_{a_1}) < 1/2 - \delta_2$. Thus the functions $A_{(h,0),(\bar{1},1)}$ and A_1 satisfy the properties required in conditions (2.1) and (2.2) of Definition 3.9.

Let $\rho \stackrel{\text{def}}{=} \max\{\rho_1, \rho_2, \rho_3\}$. We conclude that the only case which allows

$$\texttt{ACC}\left[\, V_{\text{PCPinner}}^{A,A_1}(\sigma, h)\,\right] > \rho$$

is Case 2.2.2, which also satisfies conditions (2.1) and (2.2) of Definition 3.9. Thus, V_{PCPinner} satisfies condition (2) of Definition 3.9. Clearly, V_{PCPinner} also satisfies condition (1) of Definition 3.9, and thus the lemma follows. □

6.2. The new proof system. Combining the above inner verifier with an adequate outer verifier, we obtain a pcp system for NP with query complexity 11.

THEOREM 6.2. $\mathrm{NP} = \mathrm{PCP}_{1,1/2}[\mathsf{coins} = \log\,;\ \mathsf{query} = 11\,;\ \mathsf{query}_{\mathrm{av}} = 10.89\,;\ \mathsf{free} = 7]$.

Proof. We consider a canonical (l, l_1)-inner verifier V_{PCPinner} with parameters $p_3 = 0.001$, $p_1 = 2/7 \cdot 0.999$ and $p_2 = \frac{5}{7} \cdot 0.999$. By Lemma 6.1, V_{PCPinner} is $(\rho, \delta_1, \delta_2)$-good for $\delta_1 = \delta_2 = 0.00001$ and $\rho = 0.49999$. We now choose an appropriate outer verifier. Let $\epsilon = 16 \cdot (0.5 - \rho)\delta_1^2\delta_2^2$. Lemma 3.8 provides us with l and l_1 such that an ϵ-good (l, l_1)-canonical outer verifier V_{outer} with randomness $O(\log n)$ exists. Let $V = \langle V_{\mathrm{outer}}, V_{\mathrm{PCPinner}} \rangle$ be the composition of V_{outer} and V_{PCPinner} according to the definitions in section 3.4. This verifier has randomness $O(\log n)$. Apply Theorem 3.12 to see that V has completeness parameter 1 and soundness parameter $\rho + \epsilon/(16\delta_1^2\delta_2^2) = 1/2$. The query (and free-bit) complexity of V is the same as that of V_{PCPinner} above (i.e., 11 and 7, respectively).

To obtain the bound on the average query complexity, we observe that we can afford not to perform the RMB test with some small probability. Specifically, Case 2.1 in the proof of Lemma 6.1, which is the only case where the RMB test is used, yields error of $0.48505832 + \frac{p_3}{1-p_3}$. Thus, if we modify V_{PCPinner} so that, whenever the RMB test is invoked it is performed only with probability 0.973, we get that Case 2.1 detects violation with probability at least $(1 - 0.48505832 - 0.0010011) \cdot 0.973 > 0.50006$. Consequently, the modified inner verifier errs with probability bounded away from $1/2$ and so does the composed verifier. The modification decreases the average query complexity by $(1 - 0.973) \cdot (2 + p_2 \cdot 3) > 0.027 \cdot 4.12 > 0.11$. (The reduction is both from step 2 and the second case in step 3.) The theorem follows. ∎

7. Amortized free bits and MaxClique hardness.

7.1. The iterated tests. The "iterated tests" will be used in the next section to derive a proof system for NP having amortized free-bit complexity ≈ 2. Intuitively, we will be running each of the atomic tests many times, but, to keep the free-bit count low, these will NOT be independent repetitions. Rather, following [23], we will run about $2^{O(m)}$ copies of each test in a way which is pairwise, or "almost" pairwise independent, to lower the error probability to $O(2^{-m})$. This will be done using $2m$ free bits. Specifically, we will select uniformly m functions in $\mathcal{F}_l$ (and m functions in $\mathcal{F}_{l_1}$) and invoke the atomic tests with functions resulting from all possible linear combinations of the selected functions.

7.1.1. Linearity and randomness. We begin with some observations relating stochastic and linear independence. Note that $\mathcal{L}_m$ is a subvector space of $\mathcal{F}_m$, and in particular a vector space over Σ in its own right. So we can discuss the linear independence of functions in $\mathcal{L}_m$. We say that $\vec{L} = (L_1, \ldots, L_k) \in \mathcal{L}_m^k$ is linearly independent if $L_1, \ldots, L_k$ are linearly independent. Furthermore, we say that $\vec{L}_1 = (L_{1,1}, \ldots, L_{1,k})$ and $\vec{L}_2 = (L_{2,1}, \ldots, L_{2,k})$ are *mutually linearly independent* if the $2k$ functions $L_{1,1}, L_{2,1}, \ldots, L_{1,k}, L_{2,k}$ are linearly independent.

LEMMA 7.1. *For $\vec{L} = (L_1, \ldots, L_k) \in \mathcal{L}_m^k$ let $J_{\vec{L}} \colon \mathcal{F}_l^m \to \mathcal{F}_l^k$ be defined by $J_{\vec{L}}(\vec{f}) = (L_1 \circ \vec{f}, \ldots, L_k \circ \vec{f})$, for $\vec{f} = (f_1, \ldots, f_m)$. Fix $\vec{L}$ and consider the probability space defined by having $f_1, \ldots, f_m$ be uniformly and independently distributed over $\mathcal{F}_l$. Regard the $J_{\vec{L}}$'s as random variables over the above probability space. Then*

(1) *if $\vec{L}$ is linearly independent, then $J_{\vec{L}}$ is uniformly distributed in $\mathcal{F}_l^k$.*

(2) *if $\vec{L}_1, \vec{L}_2$ are mutually linearly independent, then $J_{\vec{L}_1}$ and $J_{\vec{L}_2}$ are independently distributed.*

The iterated tests. Here A: $\mathcal{F}_l \to \Sigma$ and A_1: $\mathcal{F}_{l_1} \to \Sigma$ are the objects being tested. The tests also take additional inputs or parameters: below $\vec{f} \in \mathcal{F}_l^m$; $\vec{g} \in \mathcal{F}_{l_1}^m$; $L, L_1, L_2, L_3 \in \mathcal{L}_m$; and σ: $\Sigma^l \to \Sigma^{l_1}$. The tests are specified in terms of the atomic tests of Figure 8.

$$\mathbf{LinTest}^m(A; \vec{f}, L_1, L_2) = \mathbf{LinTest}(A; L_1 \circ \vec{f},\ L_2 \circ \vec{f}).$$

$$\mathbf{MBTest}^m(A; \vec{f}, L_1, L_2, L_3) = \mathbf{MBTest}(A; L_1 \circ \vec{f},\ L_2 \circ \vec{f},\ L_3 \circ \vec{f}).$$

$$\mathbf{ProjTest}_\sigma^m(A, A_1; \vec{f}, \vec{g}, L) = \mathbf{ProjTest}_\sigma(A, A_1; L \circ \vec{f},\ L \circ \vec{g}).$$

The passing probabilities. These are the probabilities we are interested in:

$$\textsc{LinPass}^m(A) = \Pr_{\vec{f} \stackrel{R}{\leftarrow} \mathcal{F}_l^m}\left[\forall\, L_1, L_2 \in \mathcal{L}_m\ :\ \mathbf{LinTest}^m(A; \vec{f}, L_1, L_2) = 0\right].$$

$$\textsc{MBPass}^m(A) = \Pr_{\vec{f} \stackrel{R}{\leftarrow} \mathcal{F}_l^m}\left[\forall\, L_1, L_2, L_3 \in \mathcal{L}_m\ :\ \mathbf{MBTest}^m(A; \vec{f}, L_1, L_2, L_3) = 0\right].$$

$$\textsc{ProjPass}_\sigma^m(A, A_1) = \Pr_{\vec{f} \stackrel{R}{\leftarrow} \mathcal{F}_l^m\ ;\ \vec{g} \stackrel{R}{\leftarrow} \mathcal{F}_{l_1}^m}\left[\forall\, L \in \mathcal{L}_m\ :\ \mathbf{ProjTest}_\sigma^m(A, A_1; \vec{f}, \vec{g}, L) = 0\right].$$

FIG. 15. *The iterated tests and their passing probabilities.*

The proof of this lemma is quite standard and thus omitted; it amounts to saying that linearly independent combinations of stochastically independent random variables result in stochastically independent random variables.

The analysis of the iterated projection test (see Figure 15) can be done in a relatively straightforward way, given the above, because the invoked projection test uses a single linear combination of each sequence of random functions, rather than several such combinations (as in the other iterated tests). Thus we begin with the iterated projection tests. The analysis of the other iterated tests, where the atomic tests are invoked on two/three linear combinations of the same sequence of random function, require slightly more care. The corresponding lemmas could have been proven using the notion of "weak pairwise independence" introduced in [23]. However, we present here an alternative approach.

7.1.2. Iterated projection test. The *iterated projection test* described in Figure 15 takes as input vectors $\vec{f}, \vec{g} \in \mathcal{F}_l^m$, and also a linear function $L \in \mathcal{L}_m$. Note that $f = L \circ \vec{f}$ is in $\mathcal{F}_l$, and $g = L \circ \vec{g}$ is in $\mathcal{F}_{l_1}$. Thus, the test is just the atomic projection test on f and g. The following lemma says that if the passing probability $\textsc{ProjPass}_\sigma^m(A)$, representing 2^m invocations of the atomic projection test, is even slightly significant, and if A is close to E_a, then A_1 is close to the encoding of the projection of a.

LEMMA 7.2. *There is a constant c_3 such that the following is true. Let σ: $\Sigma^l \to \Sigma^{l_1}$ be a function. Let $a \in \Sigma^l$ be such that* $\mathrm{Dist}(E_a, A) \le 1/4$, *and let $a_1 = \sigma(a) \in \Sigma^{l_1}$. If* $\textsc{ProjPass}_\sigma^m(A, A_1) \ge c_3 \cdot 2^{-m}$, *then* $\mathrm{Dist}(E_{a_1}, A_1) \le 0.1$.

Proof. The proof is similar to that of [23, Lemma 3.5]. Let $\epsilon_1 = \mathrm{Dist}(A_1, E_{a_1})$ and assume it is at least 0.1. We show that there is a constant c_3 such that $\textsc{ProjPass}_h^m(A) < c_3 \cdot 2^{-m}$.

Let $N = |\mathcal{L}_m^*| = 2^m - 1$. For $L \in \mathcal{L}_m^*$ let $X_L \colon \mathcal{F}_l^m \times \mathcal{F}_{l_1}^m \to \Sigma$ be defined by

$$X_L(\vec{f}, \vec{g}) \stackrel{\text{def}}{=} \mathbf{ProjTest}_\sigma^m(A, A_1; \vec{f}, \vec{g}, L) \;=\; \mathbf{ProjTest}_\sigma(A, A_1; L \circ \vec{f},\, L \circ \vec{g}) \,.$$

Regard it as a random variable over the uniform distribution on $\mathcal{F}_l^m \times \mathcal{F}_{l_1}^m$. Let $X = \sum_{L \in \mathcal{L}_m^*} X_L$. It suffices to show that $\Pr[\, X = 0 \,] \le O(1/N)$.

Lemma 7.1 (with $k = 1$) implies that $\{X_L\}_{L \in \mathcal{L}_m^*}$ are pairwise independent, identically distributed random variables. Let $L \in \mathcal{L}_m^*$ and let $p = \mathbf{E}[X_L]$. Again using Lemma 7.1 we have

$$\begin{aligned} p &= \Pr_{\vec{f} \stackrel{R}{\leftarrow} \mathcal{F}_l^m \,;\, \vec{g} \stackrel{R}{\leftarrow} \mathcal{F}_{l_1}^m} \left[\mathbf{ProjTest}_\sigma(A, A_1; L \circ \vec{f},\, L \circ \vec{g}) = 1 \right] \\ &= \Pr_{f \stackrel{R}{\leftarrow} \mathcal{F}_l \,;\, g \stackrel{R}{\leftarrow} \mathcal{F}_{l_1}} \left[\mathbf{ProjTest}_\sigma(A, A_1; f, g) = 1 \right] , \end{aligned}$$

but by Lemma 3.21 p is at least $\epsilon_1(1 - 2\epsilon) \ge 0.05$, since $\epsilon \stackrel{\text{def}}{=} \mathrm{Dist}(E_a, A) \le 1/4$. We can conclude by applying Chebyshev's inequality. Namely,

$$\Pr[\, X = 0 \,] \;\le\; \Pr[\, |X - Np| \ge Np \,] \;\le\; \frac{Np}{(Np)^2} \;\le\; \frac{20}{N} \,,$$

as desired. □

7.1.3. Technical claim. For analyzing the other two tests we will use the following simple claim.

CLAIM 7.3. *Let $k \ge 1$ and $N = 2^m$. Then $\mathcal{L}_m^k$ contains a subset S of cardinality $\frac{N}{2^{2k}}$ such that every $\vec{L}_1 \ne \vec{L}_2 \in S$ are mutually linearly independent.*

Proof. Let $\vec{L} \in \mathcal{L}_m^k$ be linearly independent. Then, the probability that L chosen uniformly in $\mathcal{L}_m$ is linearly independent of $\vec{L}$ is $1 - \frac{2^k}{N}$. Thus, the probability that a uniformly chosen $\vec{L}' \in \mathcal{L}_m^k$ is mutually linearly independent of $\vec{L}$ is greater than $1 - \sum_{i=1}^{k} \frac{2^{k+i-1}}{N} > 1 - \frac{2^{2k}}{N}$. Now, consider a graph with vertex set $\mathcal{L}_m^k$ and edges connecting pairs of mutually linearly independent sequences (i.e., $\vec{L}_1$ and $\vec{L}_2$ are connected if and only if they are mutually linearly independent). This graph has N^k vertices and every vertex which is linearly independent has degree greater than $(1 - \frac{2^{2k}}{N}) \cdot N^k$. Clearly this graph has a clique of size $\frac{N}{2^{2k}}$ (e.g., consider a greedy algorithm which picks a vertex of maximal degree among all vertices connected to the previously selected vertices). Noting that a clique corresponds to a set of mutually linear independent sequences, we are done. □

7.1.4. Iterated linearity test. The *iterated linearity test* described in Figure 15 takes as input a vector $\vec{f} \in \mathcal{F}_l^m$ and also linear functions $L_1, L_2 \in \mathcal{L}_m$. Note that $f_1 = L_1 \circ \vec{f}$ and $f_2 = L_2 \circ \vec{f}$ are in $\mathcal{F}_l$. The test is just the atomic linearity test on these inputs. The following lemma says that if the passing probability is even slightly significant, then A is almost linear.

LEMMA 7.4. *There is a constant c_1 such that if $\textsc{LinPass}^m(A) \ge c_1 \cdot 2^{-m}$, then $\mathrm{Dist}(A, \textsc{Lin}) \le 0.1$.*

Proof. Assume that $\epsilon \stackrel{\text{def}}{=} \mathrm{Dist}(A, \textsc{Lin}) \ge 0.1$. We show that there is a constant c_1 such that $\textsc{LinPass}^m(A) < c_1 \cdot 2^{-m}$. For $\vec{L} = (L_1, L_2) \in \mathcal{L}_m^2$ let $X_{\vec{L}} \colon \mathcal{F}_l^m \to \Sigma$ be

defined by

$$X_{\vec{L}}(\vec{f}) \stackrel{\text{def}}{=} \mathbf{LinTest}^m(A; \vec{f}, L_1, L_2) \;=\; \mathbf{LinTest}(A; L_1 \circ \vec{f},\, L_2 \circ \vec{f}) \,.$$

Regard it as a random variable over the uniform distribution on $\mathcal{F}_l^m$. Let $S \subset \mathcal{L}_m^2$ be a set as guaranteed by Claim 7.3 and $X = \sum_{\vec{L} \in S} X_{\vec{L}}$. It suffices to show that $\Pr[\, X = 0 \,] \leq O(2^{-m})$. (Thus our analysis of $\text{LinPass}^m(A)$ is based only on a small fraction of all possible invocations of the iterated linear test; yet, this small fraction corresponds to a sufficiently large number of invocations.)

Using Lemma 7.1 (with $k = 2$), it follows that the random variables $\{X_{\vec{L}}\}_{\vec{L} \in S}$ are pairwise independent and that for every $\vec{L} \in S$,

$$p \stackrel{\text{def}}{=} \Pr_{\vec{f} \stackrel{R}{\leftarrow} \mathcal{F}_l^m} \left[X_{\vec{L}}(\vec{f}) = 1 \right] \;=\; \Pr_{f_1, f_2 \stackrel{R}{\leftarrow} \mathcal{F}_l} [\mathbf{LinTest}(A; f_1, f_2) = 1] \,.$$

By Lemma 3.15, $p \geq \Gamma_{\text{lin}}(\epsilon)$, and so $p \geq 3\epsilon - 6\epsilon^2$ if $\epsilon \leq 1/4$ and $p \geq 45/128$ otherwise. In either case, for $\epsilon \geq 0.1$, we get $p > 0.2$. Now by Chebyshev's inequality we have

$$\Pr[\, X = 0 \,] \leq \Pr[\, |X - N'p| \geq N'p \,] \leq \frac{1}{N'p} < \frac{5}{N'},$$

where $N' \stackrel{\text{def}}{=} |S| = 2^m/2^{2 \cdot 2} = 2^m/16$. The lemma follows. □

7.1.5. Iterated RMB test. The *iterated respect of monomial basis test* in Figure 15 takes an input $\vec{f}$ and also takes three linear functions $L_1, L_2, L_3 \in \mathcal{L}_m$. For simplicity of exposition, we assume that A is folded over $(\bar{1}, 1)$. (This assumption is justified by our usage of the test; see next subsection.) If the probability $\text{MBPass}^m(A)$ is significant, we can conclude that the linear function close to A respects the monomial basis.

LEMMA 7.5. *There is a constant c_2 such that the following is true. Let $A\colon \mathcal{F}_l \to \Sigma$ so that $A(f + \bar{1}) = A(f) + 1$, for every $f \in \mathcal{F}_l$. Let $\epsilon \leq 0.1$ so that A is ϵ-close to a linear function $\tilde{A}$ and suppose that $\text{MBPass}^m(A) \geq c_2 \cdot 2^{-m}$. Then $\tilde{A}$ respects the monomial basis.*

Proof. Assume that $\tilde{A}$ is linear but does not respect the monomial basis. We will show that there is a constant c_2 such that $\text{MBPass}^m(A) < c_2 \cdot 2^{-m}$.

For $\vec{L} = (L_1, L_2, L_3) \in \mathcal{L}_m^3$ let $X_{\vec{L}}\colon \mathcal{F}_l^m \to \Sigma$ be defined by

$$X_{\vec{L}}(\vec{f}) \stackrel{\text{def}}{=} \mathbf{MBTest}^m(A; \vec{f}, L_1, L_2, L_3) \;=\; \mathbf{MBTest}(A; L_1 \circ \vec{f},\, L_2 \circ \vec{f},\, L_3 \circ \vec{f}) \,.$$

Regard it as a random variable over the uniform distribution on $\mathcal{F}_l^m$. Again, let $S \subset \mathcal{L}_m^3$ be a set as guaranteed by Claim 7.3 (in this case $|S| = 2^m/2^{2 \cdot 3}$) and $X = \sum_{\vec{L} \in S} X_{\vec{L}}$. It suffices to show that $\Pr[\, X = 0 \,] \leq O(2^{-m})$.

Using Lemma 7.1 (with $k = 3$), it follows that the random variables $\{X_{\vec{L}}\}_{\vec{L} \in S}$ are pairwise independent and that for every $\vec{L} \in S$

$$p \stackrel{\text{def}}{=} \Pr_{\vec{f} \stackrel{R}{\leftarrow} \mathcal{F}_l^m} \left[X_{\vec{L}}(\vec{f}) = 1 \right] \;=\; \Pr_{f_1, f_2, f_3 \stackrel{R}{\leftarrow} \mathcal{F}_l} [\mathbf{MBTest}(A; f_1, f_2, f_3) = 1] \,.$$

By Lemma 3.19, $p \geq \frac{3}{8} - 7\epsilon/4 + 5\epsilon^2/2 - \epsilon^3$. Using $\epsilon \leq 0.1$, it follows that $p > 0.2$. Using Chebyshev's inequality, as in the previous proof, we are done. □

Remark 7.6. For general A's (which are not folded over $(\bar{1}, 1)$), a similar result can be proven by augmenting the iterated RMB test so that on input A, $\vec{f}$, and $\vec{L} = (L_1, L_2, L_3)$ it also checks if $A((L_1 \circ \vec{f}) + \bar{1}) = A(L_1 \circ \vec{f}) + 1$.

7.1.6. Putting some things together. The last two lemmas above allow us to conclude that if $A_{(h,0),(\bar{1},1)}$ passes the first two tests with any significant probability, then $A_{(h,0),(\bar{1},1)}$ is close to some evaluation operator E_a so that $h(a) = 0$. Thus, again, there is no need for a "circuit test."

COROLLARY 7.7. *There is a constant c such that the following is true. Let $A\colon \mathcal{F}_l \to \Sigma$, and suppose* $\text{LINPASS}^m(A_{(h,0),(\bar{1},1)}) \geq c \cdot 2^{-m}$ *and* $\text{MBPASS}^m(A_{(h,0),(\bar{1},1)}) \geq c \cdot 2^{-m}$. *Then there is a string $a \in \Sigma^l$ such that* $\text{Dist}(E_a, A_{(h,0),(\bar{1},1)}) \leq 0.1$ *and* $h(a) = 0$.

Proof. Let c be the larger of the constants from Lemmas 7.4 and 7.5. By the first lemma there is a linear $\tilde{A}$ such that $\text{Dist}(A_{(h,0),(\bar{1},1)}, \tilde{A}) < 0.1$. Now the second lemma implies that $\tilde{A}$ respects the monomial basis (using the fact that $A_{(h,0),(\bar{1},1)}(f + \bar{1}) = A_{(h,0),(\bar{1},1)}(f) + 1$ for all f's). So Proposition 3.2 says that $\tilde{A}$ is an evaluation function. Finally, by Proposition 3.6, we have $h(a) = 0$. ∎

7.2. NP in amortized free-bit complexity 2.

SOURCES OF OUR IMPROVEMENTS. We adopt the basic framework of the construction of proof systems with low free-bit complexity as presented in [23]. Our improvement comes from the use of the new long code, instead of the Hadamard code, as a basis for the construction of inner verifiers. This allows us to save one bit in the amortized free-bit complexity, the reason being that the long code contains explicitly all functions of the encoded string, whereas the Hadamard code contains only linear combinations of the bits of the string. Typically, we need to check that the verifier accepts a string and this condition is unlikely to be expressed by a linear combination of the bits of the string. Thus, one needs to keep also the linear combinations of all two-bit products and, using these extra combinations (via self-correcting), increases the amortized free bit by one. Instead, as seen above, the long code allows us to directly handle any function. The fact that we take linear combinations of these functions should not confuse the reader; these are linear combinations of random functions rather than linear combinations of random LINEAR functions (as in [23]).

Our construction of a proof system with amortized free-bit complexity of two bits is obtained by composing the (l, l_1)-canonical outer verifier of Lemma 3.8 with a (l, l_1)-canonical inner verifier, denoted $V_{\text{free-in}}$, which is depicted in Figure 16. The inner verifier $V_{\text{free-in}}$ consists of invoking the three iterated tests of Figure 15. In addition, $V_{\text{free-in}}$ also applies the linearity test to the oracle A_1. This is not done to improve the rejection probability of $V_{\text{free-in}}$ (in case the oracles A and A_1 are far from being fine), but rather to decrease the number of accepting configurations (and consequently the free-bit complexity). We also remark that $V_{\text{free-in}}$ invokes the iterated tests while providing them with access to a double folding of A (i.e., $A_{(h,0),(\bar{1},1)}$) rather than to A itself. This eliminates the need for checking that A encodes a string which evaluates to zero under h and simplifies the iterated RMB test (see Remark 7.6 at the end of subsection 7.1.5). However, unlike in previous subsections, these simplifications do not buy us anything significant (here), since the additional testing could have been done without any additional cost in free bits.

LEMMA 7.8. *There exists a constant c such that the following is true. Let l, l_1, m be integers. Then the (l, l_1)-canonical inner verifier $V_{\text{free-in}}$ with parameter m is $(\rho, \delta_1, \delta_2)$-good, where $\rho = c \cdot 2^{-m}$ and $\delta_i = 0.4$, for $i = 1, 2$.*

Proof. Here the analysis can be less careful than in analogous statements such as in Lemmas 4.1 and 5.2. Using Corollary 7.7, with respect to the oracle $A_{(h,0),(\bar{1},1)}$, we conclude that if $A_{(h,0),(\bar{1},1)}$ passed both the iterated linearity and RMB tests with

The free inner verifier. Given functions $h \in \mathcal{F}_l$ and $\sigma\colon \Sigma^l \to \Sigma^{l_1}$, the verifier has access to oracles for $A\colon \mathcal{F}_l \to \Sigma$ and $A_1\colon \mathcal{F}_{l_1} \to \Sigma$. It also takes an integer parameter m.

Random choices: $\vec{f} \stackrel{R}{\leftarrow} \mathcal{F}_l^m$; $\vec{g} \stackrel{R}{\leftarrow} \mathcal{F}_{l_1}^m$

$\forall\, L_1, L_2 \in \mathcal{L}_m$: $\mathbf{LinTest}^m(A_{(h,0),(\bar{1},1)}; \vec{f}, L_1, L_2)$

$\forall\, L_1, L_2, L_3 \in \mathcal{L}_m$: $\mathbf{MBTest}^m(A_{(h,0),(\bar{1},1)}; \vec{f}, L_1, L_2, L_3)$

$\forall L \in \mathcal{L}_m$: $\mathbf{ProjTest}_\sigma^m(A_{(h,0),(\bar{1},1)}, A_1; \vec{f}, \vec{g}, L)$

$\forall\, L_1, L_2 \in \mathcal{L}_m$: $\mathbf{LinTest}^m(A_1; \vec{g}, L_1, L_2)$

Remark: access to $A_{(h,0),(\bar{1},1)}(f)$ is implemented by accessing either $A(f)$, $A(f+h)$, $A(f+\bar{1})$, or $A(f+h+\bar{1})$.

FIG. 16. *The free inner verifier* $V_{\text{free-in}}$.

probability at least $c \cdot 2^{-m}$, then there exists a string $a \in \Sigma^l$ such that

$$\text{Dist}(E_a, A_{(h,0),(\bar{1},1)}) \;\le\; 0.1 \;=\; \frac{1}{2} - \delta_1 < \frac{1}{4}$$

and $h(a) = 0$. Using Lemma 7.2, we conclude that if $(A_{(h,0),(\bar{1},1)}, A_1)$ passed the iterated projection test, with probability at least $c_3 \cdot 2^{-m}$, then

$$\text{Dist}(E_{\sigma(a)}, A_1) \;<\; 0.1 \;=\; \frac{1}{2} - \delta_2\ .$$

Setting $\rho = c' \cdot 2^{-m}$, where $c' = \max\{c, c_3\}$, we conclude that $V_{\text{free-in}}$ satisfies condition (2) of Definition 3.9. Clearly, $V_{\text{free-in}}$ also satisfies condition (1), and the lemma follows. □

PROPOSITION 7.9. *Let l, l_1, m be integers. Then the (l, l_1)-canonical inner verifier $V_{\text{free-in}}$ with parameter m uses $2m$ free bits.*

Proof. We consider only accepting computations of $V_{\text{free-in}}$. We start by observing that all oracle values obtained from A, during the iterated linearity test (on $A_{(h,0),(\bar{1},1)}$), are determined by the values of A in locations $f_1', f_2', \ldots, f_m'$, where each f_i' is one of the four functions $f_i, f_i+h, f_i+\bar{1}$, and $f_i+h+\bar{1}$. Likewise, all oracle values obtained from A, during the iterated RMB test, are determined by the values of A in these locations $f_1', f_2', \ldots, f_m'$. Finally, all oracle values obtained from A, during the iterated projection test, are determined by the values of A_1 in locations $L \circ \vec{g}$ (for all L's) and the values of A in the locations $f_1', f_2', \ldots, f_m'$.

Now we use the fact that $V_{\text{free-in}}$ applies an iterated linearity test to the oracle A_1. It follows that all oracle values obtained from A_1, in accepting computations of $V_{\text{free-in}}$, are determined by the values of A_1 in locations $g_1, g_2, \ldots, g_m$.

We conclude that, in accepting computations of $V_{\text{free-in}}$, all values obtained from the oracles are determined by $2m$ bits (i.e., $A(f_1'), \ldots, A(f_m')$ and $A_1(g_1), \ldots, A_1(g_m)$). □

Composing the canonical outer verifier of Lemma 3.8 and the canonical inner verifier $V_{\text{free-in}}$, we get the following.

THEOREM 7.10. *For any $\epsilon > 0$ it is the case that* $\text{NP} \subseteq \overline{\text{FPCP}}[\log, 2+\epsilon]$.

Proof. Given an NP language L and an integer m (see below), we use Lemma 3.8 to construct a 2^{-m}-good outer verifier, denoted V_{outer}, for L. Recall that this outer verifier uses logarithmic randomness (actually the randomness also depends linearly on m which is a constant). Next, compose V_{outer} with the $(c \cdot 2^{-m}, 0.4, 0.4)$-good inner verifier, $V_{\text{free-in}}$, guaranteed by Lemma 7.8, where $V_{\text{free-in}}$ uses m as its integer parameter. The composed verifier has free-bit complexity $2m$ (as inherited from $V_{\text{free-in}}$ by Proposition 7.9). By Theorem 3.12 the soundness error of the composed verifier is at most $c \cdot 2^{-m} + 2^{-m}$. Selecting m to be sufficiently large (i.e., $m \geq \frac{2+\epsilon}{\epsilon} \cdot \log_2(c+1)$), the theorem follows. ∎

7.3. Hardness of MaxClique. See section 2.4 for definitions of the MaxClique and ChromNum problems and their associated gap problems, and section 2.4.3 for a description of previous work. Using the FGLSS transformation, we get the following.

THEOREM 7.11. *For any $\epsilon > 0$,*

(1) $\text{NP} \leq_R^K \textsf{Gap-MaxClique}_{c,s}$ *for* $s(N) = N^\epsilon/N$ *and* $c(N) = N^{\frac{1}{3}}/N$.

(2) $\text{NP} \leq_D^K \textsf{Gap-MaxClique}_{c,s}$ *for* $s(N) = N^\epsilon/N$ *and* $c(N) = N^{1/4}/N$.

Proof. For part (1) we use Corollary 11.3 (below), with $r = O(\log n)$ and $k = \frac{r}{\epsilon}$. We get that NP is randomly reducible to a pcp system with randomness $r + k + O(1)$, free-bit complexity $(2+\epsilon)k$, and error probability 2^{-k}. The FGLSS graph corresponding to the resulting pcp system has size $N = 2^{(r+k+O(1))+(2+\epsilon)k}$ and a gap in clique size of factor 2^k, which can be rewritten as $N^{1/(1+2+2\epsilon)}$. The clique size, in case the input is not in the language, is 2^r which can be rewritten as N^ϵ. Substituting ϵ for $\epsilon/2$, the claim of part 1 follows. For part 2 we use Corollary 11.5 and get a pcp system for NP with randomness $r + (2+\epsilon)k$, free-bit complexity $(2+\epsilon)k$, and error probability 2^{-k}. Using the FGLSS construction on this system, the claim of part (2) follows. ∎

Combining the above with a recent reduction of Fürer [45], we get the following.

THEOREM 7.12. *For any $\epsilon > 0$,*

(1) $\text{NP} \leq_R^K \textsf{Gap-ChromNum}_{c,s}$ *for* $c(N)/s(N) = N^{\frac{1}{5}-\epsilon}$.

(2) $\text{NP} \leq_D^K \textsf{Gap-ChromNum}_{c,s}$ *for* $c(N)/s(N) = N^{\frac{1}{7}-\epsilon}$.

Part II: Proofs and approximation: Potential and limitations.

8. The reverse connection and its consequences. Feige et al. [40] describe a procedure which takes a verifier V and an input x and constructs a graph, which we denote $\mathcal{G}_V(x)$, whose vertices correspond to possible accepting transcripts in V's computation and edges corresponding to consistent/nonconflicting computations. They then show the following connection between the maximum (over all possible oracles) acceptance probability of the verifier and the clique size in the graph. Recall that $\textsf{ACC}[V(x)] = \max_\pi \Pr_R[V^\pi(x;R) = 0]$ is the maximum accepting probability. Also recall that MaxClique(G) is the maximum clique size.

THEOREM 8.1 ((the FGLSS reduction) [40]). *If, on input x, a verifier V tosses r coins, then the following relationship holds:*

$$\textsf{ACC}[V(x)] = \frac{\text{MaxClique}(\mathcal{G}_V(x))}{2^r}.$$

In this section we essentially show an inverse of their construction.

8.1. The Clique-Gap verifier. We stress that by the term *graph* we mean an undirected simple graph (i.e., no self-loops or parallel edges).

THEOREM 8.2 (clique verifier of ordinary graphs). *There exists a verifier, denoted W, of logarithmic randomness complexity, logarithmic query length, and zero free-bit complexity, that, on input an N-node graph G, satisfies*

$$\mathtt{ACC}[\,W(G)\,] = \frac{\text{MaxClique}(G)}{N}\ .$$

Furthermore, $\mathcal{G}_W(G)$ is isomorphic to G, where the isomorphism is easily computable. Lastly, given a proof/oracle π we can construct in polynomial-time a clique of size pN in G, where p is the probability that W accepts G with oracle access to π.

Proof. On input a graph G on N nodes, the verifier W works with proofs of length $\binom{N}{2} - |E(G)|$. The proof π is indexed by the edges in $\overline{G}$ (i.e., nonedges in G). For clarity we assume that the binary value $\pi(\{u,v\})$ is either u or v. This is merely a matter of encoding (i.e., consider a 1-1 mapping of the standard set of binary values, $\{0,1\}$, to the set $\{u,v\}$). On input G and access to oracle π, the verifier W acts as follows:

—picks uniformly a vertex u in the vertex set of G.

—for every $\{u,v\} \in E(\overline{G})$, the verifier W queries the oracle at $\{u,v\}$ and rejects if $\pi(\{u,v\}) \neq u$.

—if the verifier did not reject by now (i.e., all queries were answered by u), then it accepts.

Properties of W. Clearly, W tosses $\log_2 N$ coins. Also, once W picks a vertex u, the only pattern it may accepts is $(u,u,\ldots,u)$. Thus the free-bit complexity of W is 0. To analyze the probability that W accepts the input G, when given the best oracle access, we first prove the following.

CLAIM. *The graphs $\mathcal{G}_W(G)$ and G are isomorphic.*

Proof. The proof is straightforward. One needs first to choose an encoding of accepting transcripts of the computation of W on input G. We choose to use the "full transcript" in which the random coins as well as the entire sequence of queries and answers is specified. Thus, a generic accepting transcript has the form

$$T_u \stackrel{\text{def}}{=} (u, (\{u,v_1\},u), \ldots, (\{u,v_d\},u)),$$

where u is the random vertex selected by the verifier and $\{v_1,\ldots,v_d\}$ is the set of non-neighbors of u. We stress that T_u is the only accepting transcript in which the verifier has selected the vertex u. Also, for each vertex u, the transcript T_u is accepting. Thus, we may consider the 1-1 mapping, ϕ, that maps T_u to u. We claim that ϕ is an isomorphism between $\mathcal{G}_W(G)$ and G.

Suppose that T_u and T_v are adjacent in $\mathcal{G}_W(G)$. Then, by definition of the FGLSS graph, these transcripts are consistent. It follows that the same query cannot appear in both (accepting) transcripts (otherwise it would have been given conflicting answers). By definition of W, we conclude that (u,v) is NOT a nonedge, namely, $(\phi(T_u),\phi(T_v)) = (u,v) \in E(G)$. Suppose, on the other hand, that $(u,v) \in E(G)$. It follows that the query $\{u,v\}$ does not appear in either T_u or T_v. Since no other query may appear in both transcripts, we conclude that the transcripts are consistent and thus T_u and T_v are adjacent in $\mathcal{G}_G(W)$. □

By Theorem 8.1 it now follows that the probability that W accepts on input G, given the best oracle, is $\text{MaxClique}(\mathcal{G}_W(G))/N$ which by the above equals

$$\text{MaxClique}(G)/N\ .$$

Furthermore, given a proof π which makes W accept G with probability p, the accepting random strings of W constitute a clique of size pN in $\mathcal{G}_W(G)$. These accepting random strings can be found in polynomial time and they encode vertices of G (which form a clique in G). □

We now generalize the above construction to get verifiers which indicate the existence of large cliques in layered graphs. An (L, M, N)-*layered graph* is an N-vertex graph in which the vertices are arranged in L layers so that there are no edges between vertices in the same layer, and there are at most M vertices in each layer. We use a convention by which, whenever a layered graph is given to some algorithm, a partition into layers is given along with it (i.e., is implicit in the encoding of the graph).

THEOREM 8.3 (clique verifier for layered graphs). *There exists a verifier, denoted W, of logarithmic randomness complexity and logarithmic query length that, on input an (L, M, N)-layered graph G, has free-bit complexity $\log_2 M$, average free-bit complexity $\log_2(N/L)$, and satisfies*

$$\mathtt{ACC}\,[\,W(G)\,] = \text{MaxClique}(G)/L\ .$$

Furthermore, $\mathcal{G}_W(G)$ is isomorphic to G, where the isomorphism is easily computable. Lastly, given a proof/oracle π, we can construct in polynomial time a clique of size pL in G, where p is the probability that W accepts G with oracle access to π.

Proof. On input an (L, M, N)-layered graph G, the verifier W works with proofs consisting of two parts. The first part assigns every layer (i.e., every integer $i \in [L]$) a vertex in the layer (i.e., again we use a redundant encoding by which the answers are vertex names rather then an index between 1 and the number of vertices in the layer). The second part assigns to pairs of nonadjacent (in G) vertices a binary value, which again is represented as one of the two vertices. On input G and access to oracle π, the verifier W acts as follows:

—Picks uniformly a layer i in $\{1, \ldots, L\}$.
—Queries π at i and obtains as answer a vertex u. If u is not in the ith layer of G, then the verifier rejects. (Otherwise, it continues as follows.)
—For every $\{u, v\} \in E(\overline{G})$, the verifier W queries the oracle at $\{u, v\}$ and rejects if $\pi(\{u, v\}) \neq u$. (Actually, it is not necessary to query the oracle on pairs of vertices belonging to the same layer.)
—If the verifier did not reject by now (i.e., all queries were answered by u), then it accepts.

Properties of W. Here W tosses $\log_2 L$ coins. Once the first query of W is answered, specifying a vertex u, the only pattern it may accept in the remaining queries is $(u, u, \ldots, u)$. Thus, the free-bit complexity of W is $\log_2 M$, accounting for the first query which may be answered arbitrarily in $\{1, \ldots, m\}$, where $m \leq M$ is the number of vertices in the chosen layer. The average free-bit complexity is $\log_2(N/L)$ (as N/L is the average number of vertices in a layer of the graph G). Again, we can prove that $\mathcal{G}_W(G) = G$ and the theorem follows.

Proof. Here, the accepting transcripts of W, on input G, correspond to a choice of a layer i and a vertex in the ith layer (since, once a vertex is specified by the first

answer, there is only one accepting way to answer the other queries). Thus, a generic accepting transcript has the form

$$T_u \stackrel{\text{def}}{=} (i, (i, u), (\{u, v_1\}, u), \ldots, (\{u, v_d\}, u)),$$

where i is the layer selected by the verifier, u is a vertex in the ith layer of G, and $\{v_1, \ldots, v_d\}$ is the set of nonneighbors of u. Again, T_u is the only accepting transcript in which the verifier's first query is answered with vertex u, and for each vertex u, the transcript T_u is accepting. Again, we consider the 1-1 mapping ϕ that maps T_u to u and show that it is an isomorphism between $\mathcal{G}_W(G)$ and G.

Suppose that T_u and T_v are adjacent in $\mathcal{G}_G(W)$. Then, by definition of the FGLSS graph, these transcripts are consistent. We first note that u and v cannot appear in the same layer of G (otherwise the first query in the transcript would yield conflicting answers). Again, the same two-vertex query cannot appear in both (accepting) transcripts, and we conclude that $(\phi(T_u), \phi(T_v)) = (u, v) \in E(G)$. Suppose, on the other hand, that $(u, v) \in E(G)$. Clearly, u and v belong to different layers and, as before, the query (u, v) does not appear in either T_u or T_v. Since no other two-vertex query may appear in both transcripts, we conclude that the transcripts are consistent and thus T_u and T_v are adjacent in $\mathcal{G}_G(W)$. □

The theorem follows as before. □

Remark 8.4. The clique verifier W, presented in the proof of Theorem 8.3, is adaptive: the answer to its first query determines (all) of the other queries. We wonder if it is possible to construct a non-adaptive clique verifier with properties as claimed in Theorem 8.3.

8.2. Reversing the FGLSS reduction. We are interested in problems exhibiting a gap in Max-Clique size between positive and negative instances. Recall that $\overline{\text{MaxClique}}(G) = \text{MaxClique}(G)/N$ is the fraction of nodes in a maximum clique of N-node graph G. Also recall from section 2.4 that the $\mathsf{Gap}\text{-MaxClique}_{c,s}$ promise problem is (A, B), where A is the set of all graphs G with $\overline{\text{MaxClique}}(G) \geq c(N)$, and B is the set of all graphs G with $\overline{\text{MaxClique}}(G) < s(N)$. The *gap* of this problem is defined to be c/s. As a direct consequence of Theorem 8.2, we get the following.

COROLLARY 8.5. *For all functions c, s: $\mathcal{Z}^+ \to [0, 1]$ we have* $\mathsf{Gap}\text{-MaxClique}_{c,s} \in \text{FPCP}_{c,s}[\log, 0, \text{poly}]$.

The above corollary transforms the gap in the promise problem into a gap in a pcp system. However, the accepting probabilities in this pcp system are very low (also on yes instances). Below, we use Theorem 8.3 to obtain pcp systems with perfect (resp., almost-perfect) completeness for this promise problem. We start by presenting two randomized reductions of the promise problem to a layer version. Alternative methods are presented in section 11 (cf. Proposition 11.6).

PROPOSITION 8.6 (layering the clique promise problem).

(1) (obtaining a perfect layering). *There exists a polynomial-time randomized transformation, T, of graphs into layered graphs so that, on input a graph G, and integers C and L, the transformation outputs a subgraph $H = T(G, C, L)$ of G in L layers such that if* $\text{MaxClique}(G) \geq C$, *then*

$$\Pr[\,\text{MaxClique}(H) < L\,] < L \cdot 2^{-\frac{C}{2L}}.$$

Furthermore, with probability $1 - L \cdot 2^{-N/3L}$, no layer of H contains more than $2 \cdot \frac{N}{L}$ nodes.

(2) (using logarithmic randomness). *There exists a polynomial-time randomized transformation, T, of graphs into layered graphs so that, on input a graph G, and integers C and L, the transformation outputs a subgraph $H = T(G, C, L)$ of G in L layers such that if* $\mathrm{MaxClique}(G) \geq C$, *then*

$$\Pr[\,\mathrm{MaxClique}(H) \leq (1-\epsilon)\cdot L\,] < \frac{L}{\epsilon C}$$

for every $\epsilon \in [0,1]$. Furthermore, the transformation uses logarithmically many coins. Also, with probability $1 - \frac{L}{\epsilon N}$, at most ϵL layers of H contain more than $2 \cdot \frac{N}{L}$ nodes.

Proof. The *first transformation* consists of assigning to each vertex of G a randomly chosen layer of H. Namely, we construct the graph H, which is a subgraph of G, by uniformly selecting for each vertex v a layer $l(v) \in [L]$ and copying only the edges of G which connect vertices placed in different layers (of H). The construction can be carried out in random polynomial time and we show that if the original graph has a clique of size C, then with high probability the resulting graph has a clique of size L, provided $C \gg 2L\log_2 L$.

CLAIM 1. *Suppose that G has a clique of size C, denoted S. Then, the probability that all vertices in S were placed in less than L layers is at most $L \cdot 2^{-\frac{C}{2L}}$.*

Proof. We start by bounding, for each i, the probability that no vertex of S is placed in the ith layer. For each $v \in S$, we introduce the 0-1 random variable ζ_v so that $\zeta_v = 1$ if v is placed in the ith layer (i.e., $l(v) = i$) and $\zeta_v = 0$ otherwise. Let $t \stackrel{\mathrm{def}}{=} C/L$. Then, $\mathbf{E}[\sum_{v\in S}\zeta_v] = t$. Using a multiplicative Chernoff bound [75], we get

$$\begin{aligned}\Pr[\,\forall v \in S : l(v) \neq i\,] &= \Pr\left[\sum_{v\in S}\zeta_v = 0\right] \\ &< 2^{-\frac{t}{2}}.\end{aligned}$$

Call the ith layer *bad* if no vertex of S is placed in it. By the above, the probability that there exists a bad layer is smaller than $L \cdot 2^{-\frac{t}{2}}$, and the claim follows. □

It is left to bound the probability that a particular layer contains more than twice the expected number of vertices. Using again a multiplicative Chernoff bound, this probability is at most $2^{-N/3L}$ and the first part of the proposition follows.

The *second transformation* consists of selecting randomly a universal$_2$ hashing function (a.k.a., pairwise independent hash function) mapping the vertices of the graph G into the layer-set $[L]$. Namely, suppose that the function h was chosen; then we construct the graph H, which is a subgraph of G, by placing a vertex v (of G) in layer $h(v)$ of H, and copying only the edges of G which connect vertices placed in different layers (of H). The construction can be carried out in polynomial time using only logarithmic randomness (for the selection of the hashing function). We show that if the original graph has a clique of size C, then with high probability the resulting graph has a clique of size almost L, provided $L \ll C$.

CLAIM 2. *Suppose that G has a clique of size C, denoted S. Then, the probability that all vertices in S were placed in less than $(1-\epsilon)\cdot L$ layers is at most $\frac{L}{\epsilon C}$.*

Proof. Again, we bound, for each i, the probability that no vertex of S is placed in the ith layer. For each $v \in S$, we introduce the 0-1 random variable ζ_v so that $\zeta_v = 1$ if $h(v) = i$ and $\zeta_v = 0$ otherwise. Let $t \stackrel{\mathrm{def}}{=} C/L$ and $\zeta \stackrel{\mathrm{def}}{=} \sum_{v\in S}\zeta_v$. Then, $\mathbf{E}[\zeta] = t$ (which is greater than 1; otherwise the claim holds vacuously). Using the

pairwise independence of h and Chebyshev's inequality, we get

$$\begin{aligned}\Pr[\,\forall v \in S : h(v) \neq i\,] &= \Pr[\,\zeta = 0\,] \\ &\leq \frac{\mathbf{Var}[\sum_{v\in S}\zeta_v]}{t^2} \\ &< \frac{C/L}{t^2} = \frac{1}{t}.\end{aligned}$$

Call the ith layer *bad* if no vertex of S is placed in it. By the above, the expected number of bad layers is smaller than $L \cdot \frac{1}{t}$, so by Markov inequality the probability that more than ϵL layers are bad is at most $1/\epsilon t$. The claim follows. □

Again, it is left to bound the probability that a particular layer contains more than $M \stackrel{\text{def}}{=} 2N/L$. Using Chebyshev's inequality again, this probability is at most L/N. Thus, the expected number of layers having more than M vertices is at most L^2/N, and it follows that the probability that ϵL layers contain more than M vertices each is at most $\frac{L^2/N}{\epsilon L} = \frac{L}{\epsilon N}$. The second part of the proposition follows. □

Combining Theorem 8.3 and Proposition 8.6, we obtain the following. (Refer to section 2.3 for what it means for a promise problem to reduce to a complexity class.)

PROPOSITION 8.7 (reversing the FGLSS reduction, general form). *For any polynomial-time computable, positive functions* $c, s, \epsilon\colon \mathcal{Z}^+ \to [0,1]$ *we have*

(1) (randomized reduction to a pcp with perfect completeness):

$$\textsf{Gap-MaxClique}_{c,s} \leq_R^K \text{FPCP}_{1,s'}[\log, f'],$$

where $f'(N) \stackrel{\text{def}}{=} \log_2(1/c(N)) + \log_2\log_2 N + 2$ *and* $s'(N) \stackrel{\text{def}}{=} 2\log_2 N \cdot \frac{s(N)}{c(N)}$.

(2) (a pcp with almost-perfect completeness):

$$\textsf{Gap-MaxClique}_{c,s} \in \text{FPCP}_{1-4\epsilon,s'}[\log, f'],$$

where $f'(N) \stackrel{\text{def}}{=} 1 + \log_2(1/c(N)) + 2\log_2(1/\epsilon(N))$ *and* $s'(N) \stackrel{\text{def}}{=} \frac{1}{\epsilon(N)^2} \cdot \frac{s(N)}{c(N)}$.

Proof. For the *second part*, we construct a verifier for the promise problem as follows. On input an N-vertex graph G, the verifier computes $C \stackrel{\text{def}}{=} N \cdot c(N)$, $\epsilon \stackrel{\text{def}}{=} \epsilon(N)$, and $L \stackrel{\text{def}}{=} \epsilon^2 C$. It invokes the second transformation of Proposition 8.6, obtaining an (L, N, N)-layered graph $H = T(G, C, L)$. (We stress that this transformation requires only logarithmically many coin tosses.) Next, the verifier modifies H into H' by omitting (the minimum number of) vertices so that no layer of H' has more than $2N/L$ vertices. Finally, the verifier invokes the clique-verifier W of Theorem 8.3 on input H'.

The free-bit complexity of the verifier constructed above is $\log_2(2N/L) = 1 + \log_2(1/c(N)) + 2\log_2(1/\epsilon(N))$. Suppose that G is a no-instance of the promise problem. Using $\text{MaxClique}(H') \leq \text{MaxClique}(G)$ and Theorem 8.3, it follows that the constructed verifier accepts G with probability at most $\frac{\text{MaxClique}(H')}{L} \leq \frac{s(N)}{\epsilon^2(N)\cdot c(N)}$. Suppose, on the other hand, that G is a yes-instance of the promise problem. Then, with probability at least $1 - \frac{L}{\epsilon C} = 1 - \epsilon$ we have $\text{MaxClique}(H) \geq (1-\epsilon)\cdot L$, and with probability at least $1 - \frac{L}{\epsilon N} > 1 - \epsilon$ we have $\text{MaxClique}(H') \geq \text{MaxClique}(H) - \epsilon L$. Thus, with probability at least $1 - 2\epsilon$, we have $\text{MaxClique}(H') \geq (1-2\epsilon)\cdot L$. It follows that the constructed verifier, when given oracle access to an appropriate proof, accepts G with probability at least $1 - 4\epsilon$.

For the *first part*, we define a promise problem which refers to gaps in cliques of layered graphs. Specifically, we have the following.

DEFINITION. *For any function* $\ell : \mathcal{Z}^+ \to \mathcal{Z}^+$ *and* $s : \mathcal{Z}^+ \to [0,1]$*, we define the promise problem* $\mathsf{Gap}{-}\mathrm{CLG}_{\ell,s}$ *as the pair* (A, B)*, where:*

(1) A *is the set of all* $(\ell(N),\, 2N/\ell(N)\,,N)$*-layered graphs* G *with* $\mathrm{MaxClique}(G) = \ell(N)$*, and*

(2) B *is the set of all* $(\ell(N),\, 2N/\ell(N)\,,N)$*-layered graphs* G *with* $\mathrm{MaxClique}(G) < s(N) \cdot \ell(N)$*.*

The gap *of this problem is defined to be* $1/s$*.*

Using the first transformation of Proposition 8.6, we obtain $\mathsf{Gap}\text{-}\mathrm{MaxClique}_{c,s} \leq_R^K \mathsf{Gap}{-}\mathrm{CLG}_{\ell,s'}$, where $\ell(N) = \frac{c(N)\cdot N}{2\log_2 N}$ and $s'(N) = \frac{s(N)\cdot N}{\ell(N)} = 2\log_2 N \cdot \frac{s(N)}{c(n)}$. On the other hand, Theorem 8.3 asserts that $\mathsf{Gap}{-}\mathrm{CLG}_{\ell,s'} \in \mathrm{FPCP}_{1,s'}[\log, f']$, where $f'(N) \stackrel{\text{def}}{=} \log_2(2N/\ell(N))$. Observing that $f'(N) = 1 + \log_2 \frac{2\log_2 N}{c(N)}$ (which equals $\log_2(1/c(N)) + \log_2\log_2 N + 2$), the proposition follows. □

Each of the two parts of Proposition 8.7 shows that the well-known method of obtaining clique-approximation results from efficient pcp systems (cf. [40, 25, 86, 41, 23]) is "complete" in the sense that, if clique approximation can be shown NP-hard, then this can be done via this method. The precise statement is given in Theorems 8.10 and 8.11 (below). As a preparatory step, we first provide an easier-to-use form of the above proposition. The restriction that f be a constant is only for notational simplicity (as otherwise, given f as a function of $N = \|G\|$, one needs to repharse it as a function of $n = |x|$).

PROPOSITION 8.8 (reversing the FGLSS reduction, easy to use form). *Let* $f > 0$ *be a constant and let* c, s: $\mathcal{Z}^+ \to [0,1]$ *be polynomial-time computable so that*

$$\frac{c(N)}{s(N)} \geq N^{\frac{1}{1+f}}.$$

Then, for every $\epsilon > 0$*, we have*

(1) (randomized reduction to a pcp with perfect completeness):

$$\mathsf{Gap}\text{-}\mathrm{MaxClique}_{c,s} \leq_R^K \overline{\mathrm{FPCP}}[\log, f+\epsilon];$$

(2) (a pcp with almost-perfect completeness):

$$\mathsf{Gap}\text{-}\mathrm{MaxClique}_{c,s} \in \overline{\mathrm{FPCP}}_{1-o(1)}[\log, f+\epsilon].$$

Proof. We merely invoke Proposition 8.7 and calculate the amortized free-bit complexity of the resulting verifier. We may assume that $s(N) \geq 1/N$. Thus (using $c(N)/s(N) \geq N^{\frac{1}{1+f}}$), we have $c(N) \geq N^{\frac{1}{1+f}}/N = N^{\frac{-f}{1+f}}$ and $1/c(N) \leq N^{\frac{f}{1+f}}$.

For part 1, we let $\alpha(N) \stackrel{\text{def}}{=} 2\log_2 N$ and set $f'(N) \stackrel{\text{def}}{=} \log_2(1/c(N)) + \log_2 \alpha(N)$ and $s'(N) \stackrel{\text{def}}{=} \alpha(N) \cdot \frac{s(N)}{c(N)}$. By invoking Proposition 8.7 (part 1) we find that

$$\mathsf{Gap}\text{-}\mathrm{MaxClique}_{c,s} \leq_R^K \mathrm{FPCP}_{1,s'}[\log, f'],$$

and

$$\mathsf{Gap}\text{-}\mathrm{MaxClique}_{c,s} \leq_R^K \overline{\mathrm{FPCP}}[\log, \overline{f'}]$$

for $\overline{f'} = \frac{f'}{\log(1/s')}$ follows. It now remains to argue that for any $\epsilon > 0$, $\overline{f'} \leq f + \epsilon$.

Using the lower bounds on $c(N)$ and $c(N)/s(N)$, we obtain $f'(N) \leq \frac{f}{1+f}\log_2 N + \log_2 \alpha(N)$ and $\log(1/s'(N)) \geq \frac{1}{1+f}\cdot\log_2 N - \log_2\alpha(N)$. Selecting a sufficiently small $\delta > 0$ and using $\log_2\alpha(N) < \delta\cdot\log_2 N$, we get

$$\begin{aligned}\overline{f'} &\leq \frac{\frac{f}{1+f}\log_2 N + \log_2\alpha(N)}{\frac{1}{1+f}\log_2 N - \log_2\alpha(N)}\\ &< \frac{\frac{f}{1+f}+\delta}{\frac{1}{1+f}-\delta},\end{aligned}$$

and so part 1 follows. For part 2, we let α be a slowly decreasing function s.t. $\alpha(N) = o(1)$ but $\log_2(1/\alpha(N)) = o(\log N)$. We set $f'(N) \stackrel{\text{def}}{=} \log_2(1/c(N)) + 2\log_2(1/\alpha(N))$ and $s'(N) \stackrel{\text{def}}{=} \frac{1}{\alpha(N)^2}\cdot\frac{s(N)}{c(N)}$. By invoking Proposition 8.7 (part 2) we get $\textsf{Gap-MaxClique}_{c,s} \in \text{FPCP}_{1-\alpha,s'}[\log, f']$. Since $\alpha(N) = o(1)$, we conclude that

$$\textsf{Gap-MaxClique}_{c,s} \in \overline{\text{FPCP}}_{1-o(1)}[\log, \overline{f'}]$$

for $\overline{f'} = \frac{f'}{\log_2(1/s')}$. Again, it remains to argue that for any $\epsilon > 0$, $\overline{f'} \leq f + \epsilon$. Using the lower bound on $c(N)$ and $c(N)/s(N)$, we obtain $f'(N) \leq \frac{f}{1+f}\log_2 N - 2\log_2\alpha(N)$ and $\log_2(1/s'(N)) = 2\log_2\alpha(N) + \frac{1}{1+f}\log_2 N$. Selecting a sufficiently small $\delta > 0$ and using $\log_2(1/\alpha(N)) < \delta\cdot\log_2 N$, we get

$$\begin{aligned}\overline{f'} &\leq \frac{\frac{f}{1+f}\log_2 N + 2\log_2(1/\alpha(N))}{\frac{1}{1+f}\log_2 N - 2\log_2(1/\alpha(N))}\\ &< \frac{\frac{f}{1+f}+\delta}{\frac{1}{1+f}-\delta},\end{aligned}$$

and part 2 follows. □

8.3. Main consequences. Let us first state the FGLSS reduction.

THEOREM 8.9 (the FGLSS reduction, revisited). *Let $f > 0$ be a constant and $c, s\colon \mathcal{Z}^+ \to [0,1]$. Then, for every $\epsilon > 0$,*

$$\overline{\text{FPCP}}_{1-o(1)}[\log, f] \leq_R^K \textsf{Gap-MaxClique}_{c,s},$$

where $c(N)/s(N) \geq N^{1/(1+f+\epsilon)}$. Furthermore, in case the proof system is of perfect completeness, we have $c(N) = N^{-f/(1+f+\epsilon)}$ and $s(N) = N^{-(1+f)/(1+f+\epsilon)}$.

Proof. We first amplify the gap of the pcp verifier (cf. Corollary 11.3) and then apply the bare FGLSS reduction (see Theorem 8.1 and [40]) to the amplified verifier. Specifically, for any problem Π in $\overline{\text{FPCP}}[\log, f]$, we first obtain $\Pi \leq_R^K \text{FPCP}_{1,2^{-t}}[(1+\epsilon)\cdot t, f\cdot t]$, where $t(n) = \gamma\log_2 n$ (with the constant γ determined by the constant $\epsilon > 0$). The FGLSS reduction now yields a graph of size $N \stackrel{\text{def}}{=} 2^{(1+\epsilon+f)\cdot t(n)}$ with gap $2^{t(n)}$ (which can be written as $N^{\frac{1}{1+\epsilon+f}}$). Specifically, the clique size for a yes-instance (resp., no-instance) is at least $2^{(1+\epsilon)\cdot t(n)} = N^{\frac{1+\epsilon}{1+\epsilon+f}}$ (resp., at most $2^{\epsilon\cdot t(n)} = N^{\frac{\epsilon}{1+\epsilon+f}}$).

A similar procedure may be applied for any Π in $\overline{\text{FPCP}}_{1-o(1)}[\log, f]$. Specifically, by definition, for some function m, $\Pi \in \text{FPCP}_{c,2^{-m}\cdot c}[\log, m\cdot f]$, for $c(n) = 1 - o(1)$ (but we are not going to use the bound on c). Using Proposition 11.1 and Proposition 11.2 (part 2), we first obtain $\Pi \leq_R^K \text{FPCP}_{c',2^{-t}\cdot c'}[(1+\epsilon)\cdot t, f\cdot t]$, where

$c'(n) = c(n)^{t(n)/m(n)}$ and $t(n) = \gamma \log_2 n$ (with the constant γ determined by the constant $\epsilon > 0$). The FGLSS reduction now yields a graph of size $N \stackrel{\text{def}}{=} 2^{(1+\epsilon+f)\cdot t(n)}$ with gap $2^{t(n)}$ as above. □

Interestingly, the gap (for MaxClique) created by the FGLSS reduction is independent of the location of the gap in the pcp system. The main result of this section follows.

THEOREM 8.10. *Let f be a constant. Then the following statements are equivalent:*

(1) *For all $\epsilon > 0$, it is the case that* NP *reduces to* Gap-MaxClique$_{c,s}$ *with gap $c(N)/s(N) = N^{1/(1+f+\epsilon)}$.*

(2) *For all $\epsilon > 0$, it is the case that* NP *reduces to* $\overline{\text{FPCP}}[\log, f + \epsilon]$.

In both items the reduction is randomized. Furthermore, the equivalence holds for both Karp and Cook reductions.

Proof. The direction (2) ⇒ (1) follows again by Theorem 8.9. The reverse direction follows by part 1 of Proposition 8.8. □

An alternative statement is provided by the following theorem. Here the second item (existence of pcp systems with certain parameters) is weaker than in the previous theorem, but this allows the (1) ⇒ (2) direction to be proven via a deterministic reduction (instead of the randomized reduction used in the analogous proof above). Recall that $\overline{\text{FPCP}}_{1-o(1)}[\cdot, f]$ is the class of problems having a proof system with almost-perfect completeness (i.e., $c = 1 - o(1)$) and amortized free-bit complexity f.

THEOREM 8.11. *Let f be a constant. Then the following statements are equivalent:*

(1) *For all $\epsilon > 0$, it is the case that* NP *reduces to* Gap-MaxClique$_{c,s}$ *with gap $c(N)/s(N) = N^{1/(1+f+\epsilon)}$.*

(2) *For all $\epsilon > 0$, it is the case that* NP *reduces to* $\overline{\text{FPCP}}_{1-o(1)}[\log, f + \epsilon]$.

In both items the reduction is randomized and the equivalence holds for both Karp and Cook reductions. Furthermore, if item 1 *holds, with respect to deterministic reductions, so does item* 2*. Thus, if item* 1 *holds with a deterministic Karp reduction, then* NP $\subseteq \overline{\text{FPCP}}_{1-o(1)}[\log, f + \epsilon]$.

Proof. The direction (2) ⇒ (1) follows by applying Theorem 8.9. The reverse direction follows by part 2 of Proposition 8.8. □

8.4. More consequences. The equivalence between clique and FPCP described above turns out to be a useful tool in the study of the hardness of the Clique and Chromatic Number problems. Here we describe some applications. The first application is merely a rephrasing of the known reductions from the MaxClique problem to the Chromatic Number problem in a simpler and more convenient way. The remaining applications use the fact that the equivalence between FPCP and MaxClique allows us to easily shift gaps, in the MaxClique problem, from one place to another. Loosely speaking, these applications use the fact that the complexity of the promise problem Gap-MaxClique$_{c,s}$ remains unchanged when changing the parameters c and s so that the $\frac{\log_2 c(N)}{\log_2 s(N)}$ remains invariant. We stress that the ratio $c(N)/s(N)$ does not remain invariant.

REPHRASING REDUCTIONS FROM MAXCLIQUE TO CHROMATIC NUMBER. Starting with the work of Lund and Yannakakis [71], there have been several works on showing the hardness of approximating the Chromatic Number, which reduce the MaxClique problem to the Chromatic Number problem; see section 2.4.3 for a description. Yet none of these results could be stated cleanly in terms of a reduction

from MaxClique to Chromatic Number without loss of efficiency, i.e., the theorems could not be stated as saying "if approximating MaxClique to within a factor of N^α is NP-hard, then approximating Chromatic Number to within a factor of $N^{h(\alpha)}$ is NP-hard." The reason for the lack of such a statement is that these reductions use the structure of the graph produced by applying an FGLSS reduction to an FPCP result and are hence really reductions from FPCP to Chromatic Number rather than reductions from MaxClique to Chromatic Number. However, now that we know that FPCP and MaxClique are equivalent, we can go back and rephrase the old statements. Thus results of [71, 66, 23, 45] can be summarized as follows.

For every $\alpha, \epsilon, \gamma > 0$,

$$\mathsf{Gap}\text{-MaxClique}_{N^{\alpha-1},N^{\epsilon-1}} \leq_R^K \mathsf{Gap}\text{-ChromNum}_{N^{-(\epsilon+\gamma)},N^{-h(\alpha)}},$$

where

(1) $h(\alpha) = \min\{\frac{1}{6}, \frac{\alpha}{5-4\alpha}\}$ [71].
(2) $h(\alpha) = \min\{\frac{1}{11}, \frac{\alpha}{5+\alpha}\}$ [66].
(3) $h(\alpha) = \min\{\frac{1}{4}, \frac{\alpha}{3-2\alpha}\}$ [23].
(4) $h(\alpha) = \min\{\frac{1}{3}, \frac{\alpha}{2-\alpha}\}$ [45].

We note that it is an open problem whether one can get a reduction in which $h(\alpha) \to 1$ as $\alpha \to 1$. We also note that Fürer's reduction [45] is randomized while the rest are deterministic.

Reductions among MaxClique problems. Next we present an invariance of the Gap-Clique problem with respect to shifting of the gaps. The following result has also been independently observed by Feige in [37], where he uses a randomized graph product to show the result. Our description uses the properties of fpcp and its equivalence to clique approximation.

Theorem 8.12. *Let $k, \epsilon_1, \epsilon_2$ be real numbers such that $k \geq 1$ and $0 \leq \epsilon_1 < \epsilon_2 \leq 1$. Then the following hold:*

(1) $\mathsf{Gap}\text{-MaxClique}_{N^{-\epsilon_2},N^{-k\epsilon_2}} \leq_D^K \mathsf{Gap}\text{-MaxClique}_{N^{-\epsilon_1},N^{-k\epsilon_1}}$. *(Deterministic.)*
(2) $\mathsf{Gap}\text{-MaxClique}_{N^{-\epsilon_1},N^{-k\epsilon_1}} \leq_R^K \mathsf{Gap}\text{-MaxClique}_{1/2\cdot N^{-\epsilon_2},2\cdot N^{-k\epsilon_2}}$.

Proof. Part (1) is proved via a well-known graph theoretic trick. Let G be an instance of $\mathsf{Gap}\text{-MaxClique}_{N^{-\epsilon_2},N^{-k\epsilon_2}}$ with N nodes. We take the graph product of G with a complete graph on m nodes to get a graph H on $M = mN$ nodes. (By a graph product of two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ we mean a graph with vertex set $V_1 \times V_2$ where vertices (u_1, u_2) and (v_1, v_2) are connected iff $(u_i, v_i) \in E_i$ for both $i = 1, 2$.) We choose m so that if G has a clique of size $N^{1-\epsilon_2}$, then H has a clique of size $M^{1-\epsilon_1}$. Specifically, setting $m = N^{\frac{\epsilon_2-\epsilon_1}{\epsilon_1}}$, the requirement is satisfied (as a clique of size $N^{1-\epsilon_2}$ in G yields a clique of size $m \cdot N^{1-\epsilon_2} = N^{\frac{\epsilon_2-\epsilon_1}{\epsilon_1}+1-\epsilon_2} = M^{\frac{\epsilon_1}{\epsilon_2}\cdot\frac{\epsilon_2(1-\epsilon_1)}{\epsilon_1}}$ in H.) Under this choice of m we will show that if G has no cliques of size $N^{1-k\epsilon_2}$, then H has no cliques of size $M^{1-k\epsilon_1}$. This will complete the proof of part 1.

Suppose H has a clique of size $M^{1-\epsilon_1}$. Then, by construction, G must have a clique of size

$$\frac{M^{1-\epsilon_1}}{m} = \frac{N^{1-\epsilon_1}}{m^{\epsilon_1}} = N^{1-\epsilon_1-\frac{\epsilon_2-\epsilon_1}{\epsilon_1}\cdot\epsilon_1},$$

and the claim follows.

For part (2) we use the equivalence between FPCP and gaps in MaxClique and apply amplification properties of FPCP. Let $c(N) = N^{-\epsilon_1}$ and $s(N) = N^{-k\epsilon_1}$. Then,

using Corollary 8.5 (for line 1 below), Proposition 11.1 (for line 2), and part (2) of Proposition 11.2 (for line 3), we get

$$\begin{aligned}
&\mathsf{Gap}\text{-MaxClique}_{N^{-\epsilon_1},N^{-k\epsilon_1}}\\
&\quad\in\ \mathrm{FPCP}_{c,s}[\log_2 N, 0, N^2]\\
&\quad\subseteq\ \mathrm{FPCP}_{c^t,s^t}[t\cdot\log_2 N, 0, N^2]\quad\text{(for any integer constant } t\geq 1)\\
&\quad\leq_R^K\ \mathrm{FPCP}_{\frac{1}{2}\cdot c^t,2\cdot s^t}[\log_2(N^2/s^t), 0, N^2].
\end{aligned}$$

The choice of the integer t will be determined later.

Now, we go back to the Clique-Gap promised problem. Applying the FGLSS reduction to the pcp class $\mathrm{FPCP}_{\frac{1}{2}\cdot c^t,2\cdot s^t}[\log_2(N^2/s^t), 0, N^2]$, we obtain an instance of $\mathsf{Gap}\text{-MaxClique}_{\frac{1}{2}N^{-\epsilon_1 t},2N^{-k\epsilon_1 t}}$ on an M-vertex graph, where $M=\frac{N^2}{s^t}=N^{2+k\epsilon_1 t}$.

To clarify the last assertion and the rest of the proof, we introduce the notation $\mathsf{Gap}\text{-MaxClique}_{\alpha(N),\beta(N)}(N)$ which makes explicit the size parameter to which the promise problem refers. Thus, letting $\gamma\stackrel{\text{def}}{=}\frac{t}{2+tk\epsilon_1}$, we have obtained

$$\mathsf{Gap}\text{-MaxClique}_{N^{-\epsilon_1},N^{-k\epsilon_1}}(N)\ \leq_R^K\ \mathsf{Gap}\text{-MaxClique}_{\frac{1}{2}M^{-\gamma\epsilon_1},2M^{-k\cdot\gamma\epsilon_1}}(M),$$

(with M polynomial in N). Now, part 2 follows by setting t so that $\gamma=\frac{t}{2+tk\epsilon_1}\geq\frac{\epsilon_2}{\epsilon_1}$ and $t=\lceil\frac{2\epsilon_2}{(1-k\epsilon_2)\epsilon_1}\rceil$ will do. (Actually, we get $\mathsf{Gap}\text{-MaxClique}_{N^{-\epsilon_1},N^{-k\epsilon_1}}(N)\ \leq_R^K\ \mathsf{Gap}\text{-MaxClique}_{\frac{1}{2}M^{-\epsilon_2'},2M^{-k\epsilon_2'}}(M)$, for $\epsilon_2'\geq\epsilon_2$, but this can be corrected by invoking item 1.) □

The following theorem was first shown by Blum [26] using the technique of randomized graph products. Instead, we use the gap-shifting idea to show that a seemingly very weak approximator to the clique (say, $N^{1-\epsilon}$-approximation algorithm for some $\epsilon>0$) can be used to obtain a very good approximator to the clique number in graphs which are guaranteed to have very large cliques. In particular, using such an algorithm, if a graph has a clique of size $\frac{N}{k}$, then a clique of size $\frac{N}{k^{1/\epsilon}}$ can be found in such a graph in polynomial time. As observed by Blum, this can be translated into significantly better algorithms for approximate coloring of a 3-colorable graph than known currently (see item 1 in Corollary 8.15 below). Here we derive the theorem using FPCP and the gap-shifting techniques. The parameters are generalized so as to be able to conclude, say, that even if we have a $\frac{N}{2^{\sqrt{\log_2 N}}}$-approximation (for MaxClique), then we can obtain nontrivially good algorithms for 3-coloring (see item 2 in Corollary 8.15).

THEOREM 8.13. *Let $\alpha\in[0,1]$, $\beta\in[0,1/2)$, and $k>1$. Define $\epsilon:\mathcal{Z}^+\to\mathcal{R}^+$, $c\in\mathcal{R}^+$, and $g:\mathcal{Z}^+\to\mathcal{R}^+$ so that*

$$\begin{aligned}
\epsilon(N)&=\frac{\alpha}{\log_2^\beta N},\\
c&=\frac{2}{\log_2 k},\\
\text{and }\ \log_2 g(N)&=\left(\frac{c^\beta\log_2 k}{\alpha}\right)^{1/(1-\beta)}\log_2^{\beta/(1-\beta)} N.
\end{aligned}$$

Then there is a randomized $\mathrm{poly}(N^{2+c\log_2 g(N)})$-time reduction of (N-vertex) instances of

$$\mathsf{Gap}\text{-MaxClique}_{1/k,1/g}$$

to M-vertex instances of

$$\textsf{Gap-MaxClique}_{\frac{1}{2}M^{-\epsilon(M)},2M^{-1+\epsilon(M)}}.$$

Remark 8.14. Observe that $g(N) = N^{o(1)}$. Also, for $\beta = 0$ we have $\epsilon(N) = \alpha$ and $g(N) = k^{\frac{1}{\alpha}}$. Thus, the theorem states that given a $\frac{1}{4}M^{1-2\alpha}$ approximator for clique, one can solve $\textsf{Gap-MaxClique}_{1/k,1/k'}$ in polynomial time, where $k' = k^{\frac{1}{\alpha}}$.

Proof. As usual we first reduce Gap-MaxClique to FPCP and then amplify.

$$\begin{aligned}
&\textsf{Gap-MaxClique}_{1/k,1/g}\\
&\quad\in\ \text{FPCP}_{1/k,1/g}[\log_2 N, 0, N^2]\\
&\quad\subseteq\ \text{FPCP}_{(1/k)^t,(1/g)^t}[t\log_2 N, 0, N^2] \text{ (for any function } t : \mathcal{Z}^+ \to \mathcal{Z}^+.)\\
&\quad\leq_R^K\ \text{FPCP}_{\frac{1}{2}(1/k)^t,2(1/g)^t}[\log_2 N^2 g^t, 0, N^2].
\end{aligned}$$

We now show that by setting $t = c\log_2 N$ and using the FGLSS reduction, the above reduces in poly(M)-time to $\textsf{Gap-MaxClique}_{\frac{1}{2}M^{-\epsilon},2M^{-\epsilon+1}}$ in an M vertex graph, where $M = N^2 g(N)^t$.

In case the graph is a no-instance, the size of the clique is most $2(1/g(N))^t \cdot M = 2N^2$. In case the graph is a yes-instance, the clique size is at least $\frac{1}{2}(1/k)^t \cdot M$. Thus it suffices to show that $2N^2 \leq 2M^{\epsilon(M)}$ and $2k^t \leq 2M^{\epsilon(M)}$, respectively. Taking logs in both cases it suffices to show that

$$2\log_2 N \leq \epsilon(M)\log_2 M, \tag{16}$$

$$t\log_2 k \leq \epsilon(M)\log_2 M. \tag{17}$$

We first lower bound the right-hand side of both equations.

$$\begin{aligned}
\epsilon(M)\log_2 M &= \alpha \log_2^{1-\beta} M\\
&\geq \alpha \log_2^{1-\beta}(g(N)^t)\\
&\geq \alpha t^{1-\beta}\log_2^{1-\beta} g(N)\\
&= \alpha \cdot (c\log_2 N)^{1-\beta} \cdot \left(c^\beta \frac{\log_2 k}{\alpha}\log_2^\beta N\right)\\
&= c\log_2 N \log_2 k.
\end{aligned}$$

Inequality (16) now follows from the fact that $c\log_2 k = 2$. Inequality (17) follows from the fact that $t = c\log_2 N$. ∎

The following result was derived as a corollary by Blum [26] and shows the application of the above theorem to coloring graphs with low-chromatic number with relatively small number of colors. We warn the reader that the corollary does not follow directly from the above theorem; this is because it uses a Levin reduction[10] from the search version of Chromatic Number to the search version of the clique problem. However, it is possible to define search versions of all the gap problems above appropriately and verify that all the reductions work for the search problems as well (i.e., they are in fact Levin reductions). Thus the following can be derived as a corollary to the above.

[10] A Levin reduction is a polynomial-time many-to-one reduction which is augmented by corresponding polynomial-time witness transformations.

COROLLARY 8.15. *Let k be an integer.*

(1) *For $\epsilon > 0$, given an $N^{1-\epsilon}$ approximator to the clique, one can color any k-colorable graph on M nodes with $O(k^{1/\epsilon} \log M)$ colors in polynomial time.*
(2) *For $\epsilon(N) = \omega((\log N)^{-1/2})$, given an $N^{1-\epsilon(N)}$ approximator to the clique, one can color any k-colorable graph on M nodes with $M^{o(1)}$ colors in time $M^{O(\log M)}$.*

9. On the limitations of some common approaches. In this section we provide lower bounds on the free-bit complexity of two tasks which are central to all existing ("low-complexity") probabilistically checkable proofs. Specifically, we consider the task of checking that a string (given by oracle access) is "close" to a valid codeword and the task of checking that one oracle is an encoding of a projection of a string encoded by a second oracle. Here a string is considered *close* to the code if its distance from some codeword is less than half the distance of the code. Loosely speaking, we show that each of these tasks has amortized free-bit complexity of at least *one* (and this is tight by the codes and tests presented in section 7). Furthermore, we show that the amortized free-bit complexity of performing both tasks (with respect to the same given oracles) is at least *two* (and this also is tight by section 7).

Our original motivation in proving these lower bounds was to indicate that a paradigm shift is required in order to improve over our PCP systems of amortized free-bit complexity 2 (for NP). In retrospect, the paradigm shifts have amounted to the relaxation of the codeword test in [55] and to the relaxation of the projection test in [56]. Thus, our lower bounds may be considered as a justification for these (somewhat unnatural) relaxations.

In particular, the lower bound on the complexity of the codeword test relies on the particular interpretation of "closeness" used above (i.e., being at distance less than *half* the distance of the code). This requirement is not essential as can be seen in section 3.4, where we show that relaxed codeword tests, in which closeness means slightly less than the distance of the code, also suffice. Håstad's relaxation of the codeword test is different, yet it also suffices for the purpose of constructing PCP systems of amortized free-bit complexity 1 (for NP) [55]. The lower bound on the complexity of the projection test seems more robust. Yet, as shown by Håstad in [56], the projection requirements can be by-passed as well, yielding pcp systems of amortized free-bit complexity tending to 0.

9.1. The tasks. Our definitions of the various tasks/tests are quite minimal and do not necessarily suffice for PCP applications. However, as we are proving lower bounds this only makes our results stronger.

Loosely speaking, the first task consists of testing that an oracle encodes a valid codeword, or is "close" to a valid codeword, with respect to an error-correcting code of nontrivial distance (i.e., distance greater than 1). The condition regarding the distance of the code is essential since the task is easy with respect to the identity map (which is a code of distance 1). We remark that testing "closeness" to codewords with respect to codes of large distance is essential in all known pcp constructions [12, 40, 9, 8, 21, 41, 23].

The *absolute distance* between two words $w, u \in \{0,1\}^n$, denoted $\Delta(w,u)$, is the number of bits on which w and u disagree. We say that the code $E : \{0,1\}^* \mapsto \{0,1\}^*$ has *absolute distance* d if, for every m and every $x \neq y \in \{0,1\}^m$, the absolute distance between $E(x)$ and $E(y)$ is at least $d(m)$. The absolute distance between a word w and a code E, denoted $\Delta_E(w)$, is defined as the minimum absolute distance between w and a codeword of E.

DEFINITION 9.1 (codeword test). *Let $E : \{0,1\}^m \to \{0,1\}^n$ be a code of absolute distance $d > 1$. A* codeword test *(with respect to E) is an oracle machine, T, such that $T^{E(a)}(R)$ accepts for all a, R. The error probability of T is defined as the maximum accepting probability of T over oracles A of absolute distance at least $\lfloor d/2 \rfloor$ from the code E, namely,*

$$\max_{A \in \{0,1\}^n \ s.t. \ \Delta_E(A) \geq \lfloor d/2 \rfloor} \left\{ \Pr_R \left[T^A(R) \ accepts \right] \right\}.$$

(Nothing is required with respect to noncodewords which are "close" to the code.)

The second task is defined with respect to a "projection function" π and a pair of codes E_1 and E_2. Loosely speaking, the task consists of checking if the string E_1-encoded by the first oracle is mapped by π to the string that is E_2-encoded by the second oracle.

DEFINITION 9.2 (projection test). *Let $E_1 \colon \{0,1\}^m \to \{0,1\}^n$ and $E_2 \colon \{0,1\}^k \to \{0,1\}^{n'}$ be two codes and let $\pi : \{0,1\}^m \to \{0,1\}^k$ be a function. A* projection test *(with respect to the above) is a two-oracle machine, T, such that $T^{E_1(a), E_2(\pi(a))}(R)$ accepts for all a, R. The error probability of T is defined as the maximum accepting probability of T over oracle pairs $(E_1(a), E_2(b))$ where $b \neq \pi(a)$, namely,*

$$\max_{a,b \ s.t. \ \pi(a) \neq b} \left\{ \Pr_R \left[T^{E_1(a), E_2(b)}(R) \ accepts \right] \right\}.$$

(Nothing is required with respect to noncodewords.)

Finally, we consider a test T which combines the two tests above, namely, T takes two oracles A and B and performs a codeword test on A and a projection test on the pair (A, B).

DEFINITION 9.3 (combined test). *Let $E_1 \colon \{0,1\}^m \to \{0,1\}^n$ be a code of absolute distance $d > 1$, let $E_2 \colon \{0,1\}^k \to \{0,1\}^{n'}$ be two codes, and let $\pi : \{0,1\}^m \to \{0,1\}^k$ be a function. A* combined test *for (E_1, E_2, π) is a two-oracle machine T such that $T^{E_1(a), E_2(\pi(a))}(R)$ accepts on all a, R. The error probability of T is defined as the maximum accepting probability of T over oracle pairs (A, B), where either $\Delta_{E_1}(A) \geq \lfloor d/2 \rfloor$ or $A = E_1(a)$, $B = E_2(b)$ but $\pi(a) \neq b$, namely,*

$$\max_{(A,B) \in S} \left\{ \Pr_R \left[T^{A,B}(R) \ accepts \right] \right\},$$

where $S \stackrel{\text{def}}{=}$

$$\{(A,B) : (\Delta_{E_1}(A) \geq \lfloor d/2 \rfloor) \text{ or } (\exists a, b \ s.t. \ A = E_1(a) \text{ and } B = E_2(b) \text{ and } \pi(a) \neq b)\}.$$

(Nothing is required with respect to noncodeword pairs, (A, B), which are "close" to some pair $(E_1(a), E_2(b))$ with $\pi(a) \neq b$.)

Conventions and notations. The *pattern* of test T on access to oracle A (resp., oracles A and B) when using coin-sequence R consists of (R and) the sequence of queries and answers made by T. Namely, this pattern, denoted $\mathsf{pattern}_T(A; R)$ (resp., $\mathsf{pattern}_T(A, B; R)$), is defined as the sequence $(R, q_1, a_1, \ldots, q_t, a_t)$, where q_i is the ith query made by T on coin-sequence R and after receiving the answers $a_1, \ldots, a_{i-1}$. We include the queries in the pattern for sake of clarity (but they can be easily reconstructed from the coin sequence and the answers). In case T uses two oracles,

we may assume that the queries specify to which oracle they are addressed. For simplicity, we assume in the rest of this subsection that the test has access to one oracle, denoted A.

The set $\mathsf{Acc}_T(R)$ is defined to be the set of accepting patterns of T on coin-sequence R. Clearly,

$$\mathsf{Acc}_T(R) = \{\mathsf{pattern}_T(A;R) : T^A(R) \text{ accepts}\}.$$

Recall that T is said to have free-bit complexity f if for each possible coin-sequence R it holds that $|\mathsf{Acc}_T(R)| \leq 2^f$. We say that T has *average free-bit complexity* f_{av} if $\mathbf{E}_R\left[|\mathsf{Acc}_T(R)|\right] \leq 2^{f_{\text{av}}}$, when the expectation is taken uniformly over all possible coin sequences. The amortized free-bit complexity of a test is defined as $\frac{f_{\text{av}}}{\log_2(1/\epsilon)}$, where f_{av} is the average free-bit complexity of the test and ϵ is its error probability.

9.2. Lower bound for the codeword test.

PROPOSITION 9.4. *For any code of absolute distance greater than* 1*, the Codeword Test has amortized free-bit complexity of at least* $1 - o(1)$.

The amortization in the above proposition is to be understood as taking place on a fixed number of free bits whereas the length of the oracle grows. Actually, we can allow both the oracle length and the free-bit count to grow, provided that the logarithm of the number of codewords grows faster than the free-bit complexity. Alternatively, we can consider a fixed oracle length and a fix bound on the number of free bits. Actually, this is done in the following technical lemma from which the above proposition follows.

LEMMA 9.5. *Let* $E : \{0,1\}^m \mapsto \{0,1\}^n$ *be a code of absolute distance* $d > 1$*, and let* T *be a codeword test with respect to* E *having average free-bit complexity* f_{av}*. Then,* T *has error probability at least* $\max(2 - 2^{f_{\text{av}}}, \frac{1}{F} - \frac{1}{M})$*, where* $F = 2^{f_{\text{av}}}$ *and* $M = 2^m$.

In particular, if $f_{\text{av}} = 0$, then the error is 1, and for $f_{\text{av}} \geq 1$ the error is at least $\frac{1}{F} - \frac{1}{M}$.

Proof. Fix an arbitrary coin-sequence R, and let F_R denote the cardinality of the set $\mathsf{Acc}_T(R)$.

Let a_1, a_2 be selected independently and uniformly in $\{0,1\}^m$, and consider the codewords $E(a_1)$ and $E(a_2)$. With probability $\frac{1}{M}$ we have $a_1 = a_2$ and otherwise $\Delta(E(a_1), E(a_2)) \geq d$. From a_1 and a_2, we construct an oracle $A(a_1, a_2)$ as follows: if $a_1 = a_2$, then $A = E(a_1)$. Otherwise, we construct $A(a_1, a_2)$ so that it agrees with the value of the bits of both $E(a_i)$'s, whenever they are the same, and so that it is at distance $\lceil d/2 \rceil$ from $E(a_1)$. This can be done as follows: let S be the set of positions on which $E(a_1)$ and $E_2(a_2)$ disagree and let S' be a subset of S of cardinality $\lceil d/2 \rceil$. Then $A(a_1, a_2)$ equals $E(a_1)$ on all positions not in S' (and equals $E(a_2)$ on the positions in S').

We claim that, when $a_1 \neq a_2$, the oracle $A \stackrel{\text{def}}{=} A(a_1, a_2)$ is at distance at least $\lfloor d/2 \rfloor$ from the code (i.e., $\Delta_E(A) \geq \lfloor d/2 \rfloor$). This can be proved as follows: consider any $a \in \{0,1\}^m$ and observe that, by the triangle inequality,

$$\Delta(A, E(a)) \geq \Delta(E(a_1), E(a)) - \Delta(E(a_1), A) \geq d - \lceil d/2 \rceil = \lfloor d/2 \rfloor.$$

We now claim that

$$\Pr_{a_1,a_2}\left[T^{A(a_1,a_2)}(R) \text{ accepts}\right] \geq \frac{1}{F_R},$$

where the probability is taken uniformly over all possible choices of $a_1, a_2 \in \{0,1\}^m$. The key observation is that if $\mathsf{pattern}_T(E(a_1); R)$ equals $\mathsf{pattern}_T(E(a_2); R)$, then $\mathsf{pattern}_T(A(a_1, a_2); R)$ will be equal to $\mathsf{pattern}_T(E(a_1); R)$ (since no query of $T(R)$ falls in the set S, defined above). Thus, since $T^{E(a_1)}(R)$ accepts, $T^{A(a_1,a_2)}(R)$ must accept, too. This suggests to lower bound the probability that $T^{A(a_1,a_2)}(R)$ accepts by the probability that $\mathsf{pattern}_T(E(a_1); R) = \mathsf{pattern}_T(E(a_2); R)$. Consider an enumeration, $\alpha_1, \ldots, \alpha_{F_R}$, of the patterns in $\mathsf{Acc}_T(R)$ and denote by p_i the probability that $\mathsf{pattern}_T(E(a); R)$ equals the ith pattern in this enumeration, when a is uniformly selected in $\{0,1\}^m$ (i.e., $p_i \stackrel{\text{def}}{=} \Pr_a[\mathsf{pattern}_T(E(a); R) = \alpha_i]$). Thus, when a_1 and a_2 are picked at random, the probability that $\mathsf{pattern}_T(E(a_1); R) = \mathsf{pattern}_T(E(a_2); R)$ is $\sum_{i=1}^{F_R} p_i^2$. Subject to the condition $\sum_i p_i = 1$, the quantity $\sum_{i=1}^{F_R} p_i^2$ is lower bounded by $\frac{1}{F_R}$ (with an equality occurring when the p_i's are equal).

The following observations now bound the error of T:

$$\begin{aligned}
&\Pr_{a_1,a_2}\left[T^{A(a_1,a_2)}(R) \text{ accepts and } a_1 \neq a_2\right] \\
&\geq \Pr_{a_1,a_2}\left[T^{A(a_1,a_2)}(R) \text{ accepts}\right] - \Pr_{a_1,a_2}[a_1 = a_2] \\
&\geq \frac{1}{F_R} - \frac{1}{M}.
\end{aligned}$$

All of the above holds for any coin-sequence R. Now, we let R be uniformly chosen and get

$$\begin{aligned}
\Pr_{R,a_1,a_2}\left[T^{A(a_1,a_2)}(R) \text{ accepts and } a_1 \neq a_2\right] &\geq \mathbf{E}_R\left[\frac{1}{F_R}\right] - \frac{1}{M} \\
&\geq \frac{1}{F} - \frac{1}{M}.
\end{aligned}$$

(The last inequality follows by Jensen's inequality.) Thus, there must exist oracles a_1 and a_2 with $a_1 \neq a_2$ such that

$$\Pr_R\left[T^{A(a_1,a_2)}(R) \text{ accepts}\right] \geq \frac{1}{F} - \frac{1}{M},$$

but the oracle $A(a_1, a_2)$ above satisfies $\Delta_E(A(a_1, a_2)) \geq \lfloor d/2 \rfloor$, implying that the error of T is at least $\frac{1}{F} - \frac{1}{M}$.

To prove that the error is at least $2 - 2^{f_{\text{av}}}$, we observe that if $F_R = 1$ for some coin-sequence R, then $\mathsf{pattern}_T(E(a_1); R) = \mathsf{pattern}_T(E(a_2); R)$, for every two $a_1, a_2 \in \{0,1\}^m$. It follows that for every $a_1 \neq a_2$, given access to the oracle $A(a_1, a_2)$ and using coin-sequence R, the test T accepts (and is wrong in doing so). Thus, for every $a_1 \neq a_2$,

$$\Pr_R\left[T^{A(a_1,a_2)}(R) \text{ accepts}\right] \;\geq\; \Pr_R[F_R = 1] \;=\; 1 - \Pr_R[F_R > 1],$$

and the error bound follows by using $\Pr_R[F_R - 1 > 0] \leq \mathbf{E}_R[F_R - 1] = F - 1$. □

Proof of Proposition 9.4. Let T be a test for the code $E : \{0,1\}^* \to \{0,1\}^*$ so that E maps m-bit strings into $n(m)$-bit strings. Suppose that T has average free-bit complexity $f(m)$ and error $\epsilon(m)$, as a function of m (the length of strings encoded by the oracle). We first assume that $f(m) \geq 1$. Using Lemma 9.5 (and letting

$\rho(m) \stackrel{\text{def}}{=} 2^{f(m)-m}$), we lower bound the amortized free-bit complexity of T as follows:

$$\begin{aligned}\frac{f(m)}{\log_2(1/\epsilon(m))} &\geq \frac{f(m)}{-\log_2(1/2^{f(m)} - 1/2^m)} \\ &= \frac{f(m)}{f(m) - \log_2(1-\rho(m))} \\ &> \frac{f(m)}{f(m)+\rho(m)} \\ &> 1 - \rho(m).\end{aligned}$$

(For the last inequality, we have assumed $f(m) \geq 1$.) Thus, for this case, the proposition follows by our convention that the number of codewords (denoted 2^m) grows faster than exponential in the free-bit complexity $f(m)$ (i.e., $\rho(m) = \frac{2^{f(m)}}{2^m} \to 0$ with $n \to \infty$). Finally, we need to address the case in which $f(m) \geq 1$ does not hold. We consider two subcases. In the first subcase, we assume that $f(m) \to 0$ for some infinite subsequence of the m's. For these m's, we use the assertion of Lemma 9.4 that $\epsilon(m) \geq 2 - 2^{f(m)}$. Setting $g(m) \stackrel{\text{def}}{=} 2^{f(m)} - 1$, we lower bound the amortized free-bit complexity by

$$\begin{aligned}\frac{f(m)}{\log_2(1/\epsilon(m))} &\geq \frac{\log_2(1+g(m))}{-\log_2(1-g(m))} \\ &\to \frac{g(m)}{g(m)}.\end{aligned}$$

For the other subcase, we have $f(m) \geq t$, for some constant $t > 0$. Applying T for t times we get a test T' with average free-bit complexity $t \cdot f(m) \geq 1$ and error $\epsilon'(m) = \epsilon(m)^t$, which maintains the amortized free-bit complexity of T (since $\frac{f(m)}{-\log_2 \epsilon(m)} = \frac{t \cdot f(m)}{-\log_2 \epsilon'(m)}$). Applying the above analysis to T', the proposition follows. □

9.3. Lower bound for the projection test. A *projection function* is a function $\pi : \{0,1\}^* \mapsto \{0,1\}^*$ having the property that for every m there exists a k so that π maps $\{0,1\}^m$ onto $\{0,1\}^k$.

PROPOSITION 9.6. *For any pair of codes used in the two oracles and any projection function, the Projection Test has amortized free-bit complexity of at least* $1 - o(1)$.

Again, the proposition is proved by a technical lemma. Actually, the lemma refers to any function $\pi : \{0,1\}^m \mapsto \{0,1\}^k$, and its conclusion depends on the cardinality of the range of π (which in case of a projection function equals 2^k). Abusing notations we let $\pi(S) \stackrel{\text{def}}{=} \{\pi(a) : a \in S\}$.

LEMMA 9.7. *Let* $E_1 : \{0,1\}^m \mapsto \{0,1\}^n$, $E_2 : \{0,1\}^k \mapsto \{0,1\}^{n'}$ *and* $\pi : \{0,1\}^m \mapsto \{0,1\}^k$ *be as in Definition* 9.2, *and let* T *be a projection test with respect to them having average free-bit complexity* f_{av}. *Then,* T *has error probability at least* $\frac{1}{F} - \frac{1}{K}$, *where* $K = |\pi(\{0,1\}^m)|$ *and* $F = 2^{f_{\text{av}}}$. *Furthermore, if* $K > 1$, *then* T *has error probability at least* $2 - 2^{f_{\text{av}}}$.

Proof. Fixing an arbitrary coin-sequence R, let $F_R \stackrel{\text{def}}{=} |\{\mathsf{Acc}_T(R)\}|$. We consider the behavior of the test T when given oracle access to a pair of randomly and independently selected codewords. Specifically, let $S \subset \{0,1\}^m$ be a set of K strings such that for every $b \in \pi(\{0,1\}^m)$ there exists an $a \in S$ satisfying $\pi(a) = b$. We consider the behavior of T when given access to the oracles $E_1(a)$ and $E_2(\pi(a'))$, where a and a' are independently and uniformly selected in S. With probability $\frac{1}{K}$, we have

$\pi(a) = \pi(a')$. On the other hand we claim that, given access to such pair of random oracles, T accepts with probability at least $\frac{1}{F_R}$. Once the claim is proven, the lemma follows (as in the proof of the previous lemma).

Consider the set of all F_R possible accepting patterns of T on access to oracles, $E_1(a)$ and $E_2(\pi(a))$, where $a \in S$. Each such pattern consists of a pair (α, β), where α (resp., β) denotes the transcript of the test's interaction with $E_1(a)$ (resp., $E_2(\pi(a))$). Enumerating all possible F_R patterns, we denote by p_i the probability that the ith pattern occurs, when T is given access to the oracle-pair $(E_1(a), E_2(\pi(a)))$, where a is uniformly selected in S. Namely,

$$p_i \stackrel{\text{def}}{=} \Pr_{a \in S}\left[\mathsf{pattern}_T(E_1(a), E_2(\pi(a)); R) = (\alpha_i, \beta_i)\right],$$

where (α_i, β_i) is the ith accepting pattern for $T(R)$. Clearly,

(18)

$$\Pr_{a,a' \in S}\left[\mathsf{pattern}_T(E_1(a), E_2(\pi(a)); R) = \mathsf{pattern}_T(E_1(a'), E_2(\pi(a')); R) = (\alpha_i, \beta_i)\right] = p_i^2.$$

We now claim that the probability that a pair of independently chosen random oracles (i.e., $(E_1(a), E_2(b))$ selected by uniformly selecting $a, a' \in S$ and setting $b = \pi(a')$) leads to the ith pattern is at least p_i^2, namely,

$$\Pr_{a,a' \in S}\left[\mathsf{pattern}_T(E_1(a), E_2(\pi(a')); R) = (\alpha_i, \beta_i)\right] \geq p_i^2. \tag{19}$$

Equation (19) is proven by a cut-and-paste argument. Suppose

$$\mathsf{p} \stackrel{\text{def}}{=} \mathsf{pattern}_T(E_1(a), E_2(\pi(a)); R) \;=\; \mathsf{p}' \stackrel{\text{def}}{=} \mathsf{pattern}_T(E_1(a'), E_2(\pi(a')); R)$$

and consider a computation of $T^{E_1(a), E_2(\pi(a'))}(R)$. Proceeding by induction, and assuming that the first t queries are answered as in p, we show that the $t+1$st query (in our "hybrid" computation) is identical to the $t+1$st query in $\mathsf{p} = \mathsf{p}'$. If this query is directed to the first oracle, then it is answered by $E_1(a)$ (as in p) and otherwise it is answered by $E_2(\pi(a'))$ (as in p'). In both cases the answer matches the $t+1$st answer in $\mathsf{p} = \mathsf{p}'$. We conclude that whenever $\mathsf{p} = \mathsf{p}'$, the computation of $T^{E_1(a), E_2(\pi(a'))}(R)$ encounters the same pattern (p). Thus, the probability that the computation of $T^{E_1(a), E_2(\pi(a'))}(R)$ encounters the ith pattern is lower bounded by the expression in Eq. (18), and Eq. (19) follows. (We remark that for nonadaptive tests, the probability that the ith pattern is encountered equals $\sum_{i=1}^{F_R} p_i' p_i''$, where p_i' (resp., p_i'') is the sum of all p_j's satisfying $\alpha_j = \alpha_i$ (resp., $\beta_j = \beta_i$). Actually, the same holds for any test which selects its queries for each oracle independently of answers obtained from the other oracle.)

Using Eq. (19), we get

$$\begin{aligned}\Pr_{a,a' \in S}\left[\mathsf{pattern}_T(E_1(a), E_2(\pi(a')); R) \in \mathsf{Acc}_T(R)\right] &\geq \sum_{i=1}^{F_R} p_i^2 \\ &\geq \frac{1}{F_R},\end{aligned}$$

and the main part of the lemma follows. Again, the "furthermore" part follows by observing that for $F_R = 1$, $\mathsf{pattern}_T(E_1(a), E_2(\pi(a)); R) = \mathsf{pattern}_T(E_1(a'), E_2(\pi(a')); R)$, for every two $a, a' \in \{0,1\}^m$. Again this implies that, for every $a_1 \neq a_2$, given access to the oracle-pair $(E_1(a), E_2(\pi(a')))$ and using coin-sequence R, the test T (wrongly) accepts. □

9.4. Lower bound for the combined test.

PROPOSITION 9.8. *For any pair of codes used in the two oracles, so that the first code has absolute distance greater than* 1*, and for any projection function, the Combined Test has amortized free-bit complexity of at least* $2 - o(1)$.

Again, the proposition is proved by a technical lemma. Loosely speaking, the lemma asserts that a combined test of free-bit complexity $2f$ must have error probability at least $\frac{1}{8} \cdot 2^{-f}$. The lower bound extends to the case where $2f$ is a bound on the average free-bit complexity; the error probability in this case can be lower bounded by $\frac{3}{64} \cdot 2^{-f}$; see details below. It follows that the amortized free-bit complexity of such a test must be at least $\frac{2f}{f+5} \approx 2$ (for large f's). The restriction to large f's does not really weaken the result. Suppose on the contrary that there exists a test with amortized free-bit complexity f_{am}. Then, for any sufficiently large t, we can obtain a test with free-bit complexity $2f \stackrel{\text{def}}{=} t \cdot f_{\text{am}}$ and error 2^{-t}. By the above, $\frac{t \cdot f_{\text{am}}}{t} \geq \frac{2f}{f+5} \approx 2$ (as f is now large).

LEMMA 9.9. *Let* $E_1 : \{0,1\}^m \mapsto \{0,1\}^n$ *be a code of absolute distance greater than* 1*,* $E_2 : \{0,1\}^k \mapsto \{0,1\}^{n'}$*, and let* $\pi : \{0,1\}^m \mapsto \{0,1\}^k$ *be a projection function. Suppose that* T *is a combined codeword and projection test with respect to the above having free-bit complexity* $2f$*. Then,* T *has error probability at least* $\frac{1}{8F} - \frac{1}{2K} - \frac{1}{4M}$*, where* $K = 2^k$*,* $F = 2^f$*, and* M *is the minimum, over all* $b \in \{0,1\}^k$*, of the number of* $a \in \{0,1\}^m$ *projected by* π *to* b *(i.e.,* $M \stackrel{\text{def}}{=} \min_{b \in \{0,1\}^k}\{|\{a : \pi(a) = b\}|\}$*). Furthermore, if* $2f < 1$ *and* $\max\{M, K\} > 1$*, then* T *has error probability* 1*.*

Proof. The "furthermore" part follows immediately by any of the furthermore parts of Lemma 9.5 or 9.7 (as 2^{2f} must be an integer and so $2f < 1$ implies $f = 0$). The proof of the main part of the lemma uses both strategies employed in the proofs of Lemmas 9.5 and 9.7. We consider two cases. The first case is that for some $E_2(b)$, half of the possible (coin-sequence) R's have at most F accepting patterns with respect to the coin-sequence R and second oracle $B = E_2(b)$. In this case we employ the strategy used in the proof of Lemma 9.5, restricted to oracles constructed by combining two uniformly selected codewords $E_1(a_i)$'s satisfying $\pi(a_i) = b$. The second case is that for every $b \in \{0,1\}^k$, for half of the possible (coin-sequence) R's, the number of accepting patterns with respect to the coin-sequence R and second oracle $B = E_2(b)$ is at least F. In this case we show that many possible B's must fit into fewer than $\frac{F^2}{F}$ accepting patterns and we may employ the strategy used in the proof of Lemma 9.7. Details follow.

In the following, $\delta \in [0,1]$ is a constant to be determined later. (In the above motivating discussion we have used $\delta = 1/2$, but a better bound follows by letting δ be larger.)

Case 1. There exists $b \in \{0,1\}^k$ so that for at least $(1-\delta)$ fraction of the possible (coin-sequence) R's, hereafter called *good*, the number of accepting patterns with respect to the coin-sequence R and second oracle (fixed to) $B = E_2(b)$ is at most F.

Fixing this b, we consider M possible a's satisfying $\pi(a) = b$. Employing the argument of Lemma 9.5, we get that for each of these *good* R's, a random oracle A (constructed using two uniformly chosen a's as above) is wrongly accepted with probability at least $\frac{1}{F} - \frac{1}{M}$. By an averaging argument, it follows that there exists a pair of oracles (A, B) on which T errs with probability at least

$$(1-\delta) \cdot \left(\frac{1}{F} - \frac{1}{M}\right). \tag{20}$$

Case 2. For every $b \in \{0,1\}^k$, for at least a δ fraction of the possible (coin-sequence) R's, the number of accepting patterns with respect to the coin-sequence R and second oracle $B = E_2(b)$ is at least F.

Let $\gamma < \delta$ be a parameter to be determined later. By a counting argument, for at least a $\frac{\delta-\gamma}{1-\gamma}$ fraction of the possible R's, hereafter called *good*, there exists a set, denoted Π_R, of at least $\gamma \cdot 2^k$ possible $b \in \{0,1\}^k$ so that there are at least F accepting patterns which are consistent with coin-sequence R and second oracle fixed to $B = E_2(b)$. (Namely, let g denote the fraction of good R's. Then $g + (1-g)\cdot\gamma \geq \delta$ and $g \geq \frac{\delta-\gamma}{1-\gamma}$ follows.)

Let $S \subset \{0,1\}^m$ be a set of 2^k strings, defined as in the proof of Lemma 9.7, so that π maps S onto $\{0,1\}^k$. Fixing a good coin-sequence R, we adapt the strategy used in the proof of Lemma 9.7 as follows. We consider a set $S_R \subseteq S$ of $|\Pi_R|$ strings so that π maps S_R onto Π_R, and enumerate the accepting patterns which occur when the test, using coins R, is given access to an oracle-pair $(E_1(a), E_2(\pi(a)))$, where a is uniformly chosen in S_R. We first claim that there are at most F such patterns. Namely, we have the following.

CLAIM. *For any good R,* $|\{\mathsf{pattern}_T(E_1(a), E_2(\pi(a)); R) : a \in S_R\}| \leq F$.

Proof. By definition of Π_R, for each $b \in \Pi_R$, there are at least F accepting patterns consistent with the coin-sequence R and the second-oracle $E_2(b)$ (and out of them only one fits the first oracle $E_1(a)$, where $a \in S_R$ and $\pi(a) = b$). By a cut-and-paste argument, if (R, α, β) and (R, α', β) are accepting patterns for second-oracle $E_2(b)$, and if (R, α, β) is an accepting pattern for second-oracle $E_2(b')$, then (R, α', β) is also an accepting pattern for second-oracle $E_2(b')$. It follows that the accepting patterns of two $E_2(b)$'s either collide or do not intersect. Thus, the number of accepting patterns for the various $(E_1(a), E_2(\pi(a)))$'s, where $a \in S_R$, is at most $\frac{F^2}{F} = F$ and the claim follows. □

Now we consider what happens if one selects independently and uniformly $a, a' \in S$. Following the proof of Lemma 9.7, with probability $\frac{1}{K}$, we have $\pi(a) = \pi(a')$ (and otherwise $\pi(a) \neq \pi(a')$). On the other hand, given access to such a pair of random oracles, the test accepts with probability at least $\gamma^2 \cdot \frac{1}{F}$. (The γ^2 factor is due to the probability that $a, a' \in S_R$, whereas the $\frac{1}{F}$ factor corresponds to the analysis which supposes that a and a' are uniformly selected in S_R.)

The above analysis holds for any good coin-sequence R. Using the lower bound on the fraction of good R's, it follows that for a $\frac{\delta-\gamma}{1-\gamma}$ fraction of the R's, the probability that the test errs, on coin-sequence R when given access to a random pair of oracles (selected as above), is at least $\frac{\gamma^2}{F} - \frac{1}{K}$. By an averaging argument, there exists a pair of oracles for which the test errs with probability

$$\frac{\delta-\gamma}{1-\gamma} \cdot \left(\frac{\gamma^2}{F} - \frac{1}{K}\right). \tag{21}$$

Setting $\delta = \frac{3}{4}$ and $\gamma = 1/2$ we lower bound the expressions in Eqs. (20) and (21) by $\frac{1}{4F} - \frac{1}{4M}$ and $\frac{1}{8F} - \frac{1}{2K}$, respectively, and the lemma follows. □

To prove a bound for the case of average free-bit complexity $2f$, we first apply Markov's inequality and conclude that all but an ϵ fraction of the coin sequences have at most $G^2 \stackrel{\text{def}}{=} \frac{F^2}{\epsilon}$ accepting patterns (in which this fixed coin-sequence appears). (We can use any $0 < \epsilon < 1$.) We then consider only those coin sequences (and apply the same argument as above to each of them). The averaging argument at the end of the above proof then yields that there exists an oracle pair on which T errs on at least a $\frac{1}{8G} - \frac{1}{2K} - \frac{1}{4M}$ fraction of these coin sequences. It follows that this oracle makes T err with probability at least $(1-\epsilon)\cdot(\frac{1}{8G} - \frac{1}{2K} - \frac{1}{4M})$ (which equals $(1-\epsilon)\cdot(\frac{\sqrt{\epsilon}}{8F} - \frac{1}{2K} - \frac{1}{4M})$). Using $\epsilon = \frac{1}{4}$, we get a lower bound of $\frac{3}{64F} - \frac{3}{8K} - \frac{3}{16M}$.

Part III: PCP: Properties and transformations.

10. The complexity of PCP and FPCP. In this section we present several results regarding the complexity of languages acceptable by probabilistically checkable proofs having, respectively, small query complexity, small amortized query complexity, and small free-bit complexity. Thus, in the current section, notations such as $\mathrm{PCP}_{c,s}[r,q]$ stand for classes of languages. The results can be extended to classes of promise problems having such probabilistically checkable proofs.

In this section, $\mathrm{MIP}_{c,s}[r,p]$ denotes the class of languages accepted by a (one-round) p-prover interactive proof system in which r is the randomness complexity, c is a lower bound on the probability of accepting yes-instances, and s is an upper bound on the probability of accepting no-instances. The corresponding class for probabilistically checkable proofs is $\mathrm{PCP}_{c,s}[r,q]$, where q denotes the number of queries. In both classes only binary queries are allowed (indeed this is less standard for MIP).

10.1. MIP versus PCP. The first part of the following lemma is folklore and is stated here for sake of completeness.

LEMMA 10.1. *For all admissible functions c, s, r, p,*

(1) $\mathrm{MIP}_{c,s}[r,p] \subseteq \mathrm{PCP}_{c,s}[r,p]$.

(2) $\mathrm{MIP}_{c,s}[r,p] \subseteq \mathrm{MIP}_{c,2s}[r,p-1]$.

Proof. Part (1) follows from the definition of PCP and MIP. Part (2) is shown as follows. Let V be an (r,p)-restricted MIP verifier. We define V', an $(r,p-1)$-restricted verifier who on input x behaves as follows:

- V' tosses coins $\overline{c}$ for V.
- V' refers the first $p-1$ queries of V to the corresponding $p-1$ provers obtaining answers (bits) $a_1,\ldots,a_{p-1}$, respectively.
- V' accepts if and only if there exists $a_p \in \{0,1\}$ such that V would accept answers $a_1,\ldots,a_p$ on input x and random string $\overline{c}$.

Suppose that provers $P_1,\ldots,P_p$ convince V to accept x with probability δ. Then, the provers $P_1,\ldots,P_{p-1}$ convince V' to accept x with probability at least δ (because if $V(x)$ accepts the transcript $(\overline{c},a_1,\ldots,a_p)$ then $V'(x)$ will accept the transcript $(\overline{c},a_1,\ldots,a_{p-1})$). This justifies the bound on the completeness probability of V'. Suppose, on the other hand, that provers $P_1,\ldots,P_{p-1}$ cause V' to accept x with probability δ. Consider a uniformly selected strategy for another prover, denoted P_p (i.e., choose a random response for every question). Then, the probability that provers $P_1,\ldots,P_p$ cause V to accept input x is at least $\frac{1}{2}\cdot\delta$ (because if $V'(x)$ accepts the transcript $(\overline{c},a_1,\ldots,a_{p-1})$, then there exists a value $a_p \in \{0,1\}$ so that $V(x)$ will accept the transcript $(\overline{c},a_1,\ldots,a_p)$, and with probability one-half P_p's answer equals this a_p). This justifies the bound on the soundness probability of V'. □

Containments of PCP systems in MIP systems are more problematic. The reader is referred to a paper by Ta-Shma [83]. That paper also contains a proof of the following result due to Bellare, Goldreich, and Safra:

$$\mathrm{PCP}_{c,s}(\log,q) \subseteq \mathrm{MIP}_{c,q^q\cdot s}(\log,q).$$

Here we only consider the nonadaptive case and obtain a different bound on the soundness parameter.

PROPOSITION 10.2. *Suppose $L \in \mathrm{PCP}_{c,s}(r,q)$ with a nonadaptive verifier. Then $L \in \mathrm{MIP}_{c',s'}(r+O(\log q),q)$, where $c' = c + p\cdot(1-c)$, $s' = s + p\cdot(1-s)$ for any $p \geq \lfloor q/2\rfloor/(1+\lfloor q/2\rfloor)$.*

For $q=3$, we may set $p=0.5$ and obtain $c' = (c+1)/2$ and $s' = (s+1)/2$.

Proof. We start with a nonadaptive PCP verifier of q queries and construct a q-prover system as follows. First we uniformly select coin tosses for the PCP verifier, which defines q queries (here is where we use nonadaptivity). Next,

- with probability p we select a query uniformly among these q queries and forward it to all q provers. We accept iff all provers answer in the same manner.
- with probability $1-p$ we simulate the PCP system as follows. We uniformly select $i \in [q]$ and refer the jth query of the verifier to the $(i+j)$th prover. We accept iff the PCP verifier would have accepted.

Clearly, by setting all MIP provers to equal the good oracle (of the PCP system), inputs in the language are accepted with probability at least $p \cdot 1 + (1-p) \cdot c = c + p \cdot (1-c)$.

We now bound the acceptance probability of the MIP system for an input not in the language. Fix an arbitrary sequence of MIP provers. Let δ denote the probability, taken over the queries selected by the PCP verifier as above, that the q MIP provers differ on a random query. Define an oracle so that on each query it equals the majority of the prover's answers (ties, in case of even q, are broken arbitrarily). Then, the probability that the MIP system accepts is bounded above by

$$p \cdot (1-\delta) + (1-p) \cdot (s + \lfloor q/2 \rfloor \cdot \delta)\,. \tag{22}$$

To justify the second term consider the simulation of the PCP system (which takes place with probability $1-p$). In case the answers given by all MIP provers equal the corresponding answers of the PCP oracle (defined above), we bound the acceptance probability by soundness of the PCP system. Otherwise, there must be a query on which the relevant MIP prover differs from the PCP oracle. For each query this happens with probability at most $\frac{\lfloor q/2 \rfloor}{q}$ (as, by definition, only a minority of provers differ from the oracle). Using the union bound, Eq. (22) follows. Using the definition of p, we have

$$\begin{aligned} p \cdot (1-\delta) + (1-p) \cdot (s + \lfloor q/2 \rfloor \cdot \delta) &= p + (1-p) \cdot s - \delta \cdot (p - (1-p) \cdot \lfloor q/2 \rfloor) \\ &\leq s + p \cdot (1-s), \end{aligned}$$

and the proposition follows. ∎

10.2. Query complexity and amortized query complexity. The following proposition explores the limitations of probabilistically checkable proof systems which use logarithmic randomness and up to three queries. Some of the qualitative assertions are well known; for example, when considering perfect completeness, three queries are the minimum needed (and sufficient [8]) to get above P.

PROPOSITION 10.3 (PCP systems with logarithmic randomness and at most three queries).

(1) (PCP with one query is weak). *For all admissible functions* $s, c : \mathcal{Z}^+ \to [0,1]$, *so that* s *is strictly smaller than* c, $\mathrm{PCP}_{c,s}[\log, 1] = \mathrm{P}$.
(2) (one-sided error pcp with two queries is weak). *For all admissible functions* $s : \mathcal{Z}^+ \to [0,1]$ *strictly less than* 1, $\mathrm{PCP}_{1,s}[\log, 2] = \mathrm{P}$.
(3) (two-sided error pcp with two queries is not weak). *There exists* $0 < s < c < 1$ *so that* $\mathrm{PCP}_{c,s}[\log, 2] = \mathrm{NP}$. *Furthermore, this holds for some* $c > 0.9$ *and* $s < \frac{73}{74}c$.
(4) (one-sided error pcp with three queries is not weak). $\mathrm{PCP}_{1,0.85+\epsilon}[\log, 3] = \mathrm{NP}$, $\forall \epsilon > 0$.

(5) (one-sided error pcp with three queries is not very strong). *For every* $s < 0.18$, $\mathrm{PCP}_{1,s}[\log, 3] = \mathrm{P}$. *Furthermore,* $\forall s \leq 0.299$, $\mathbf{na}\mathrm{PCP}_{1,s}[\log, 3] = \mathrm{P}$, *where* $\mathbf{na}\mathrm{PCP}$ *is a restriction of* PCP *in which the verifier is required to be nonadaptive.*

We remark that $\mathrm{PCP}_{1,0.8999}[\log, 3] = \mathrm{NP}$ with a nonadaptive verifier was presented in an earlier version of this paper [20]. Using Proposition 10.2, we have $\mathrm{MIP}_{1,0.95}[\log, 3] = \mathrm{NP}$.

Proof of Proposition 10.3, *part* 1. An oracle π maximizing the acceptance probability can be constructed by scanning all possible random pads (random strings) and setting $\pi(q)$ so that it "satisfies" the majority of random pads for which the verifier makes query q. □

Proof of Proposition 10.3, *part* 2. The folklore proof commonly deals only with the nonadaptive case. In general, the verifier V, demonstrating that $L \in \mathrm{PCP}_{1,s}[\log, 2]$, may be adaptive. We assume, without loss of generality, that V always makes at least one query. Thus, after making the first query, V decides whether to accept, reject, or make an additional query and accept only a specific answer for it. Thus, the computation of V on input x, random pad $\overline{c}$, and access to a generic oracle can be captured by two Horn clauses, each corresponding to a different answer value for the first query. Specifically, suppose that V queries the oracle at location i and upon receiving value σ accepts iff location j has value τ. Then we write the Horn clause $\pi_i^\sigma \rightarrow \pi_j^\tau$, where π_i^τ is a boolean variable representing the event that ith oracle location has value τ. (In case V always accepts (resp., rejects) after obtaining value σ from oracle location i, we write the clause $\pi_i^\sigma \rightarrow \mathbf{T}$ (resp., $\pi_i^\sigma \rightarrow \mathbf{F}$).) In addition, for every i, we write the Horn clauses $\pi_i^0 \rightarrow (\neg\pi_i^1)$ and $(\neg\pi_i^0) \rightarrow \pi_i^1$. Thus, the computation of V on input x and access to a generic oracle can be captured by a Horn formula, denoted ϕ_x, in which Horn clauses correspond to the various (polynomially many) possible (random pad, first-answer) pairs. Furthermore, ϕ_x can be constructed in polynomial time given x (and V). Using a (polynomial-time) decision procedure for satisfiability of Horn formulae, we are done. (Alternatively, we can use the linear-time decision procedure for 2-SAT due to Even, Itai, and Shamir [35].) □

Proof of Proposition 10.3, *part* 4. To see that $\mathrm{PCP}_{1,s}[\log, \mathrm{poly}] \subseteq \mathrm{NP}$, for every $s < 1$, consider a nondeterministic machine which tries to guess an oracle which makes the verifier (of the above system) always accept. The other direction (of part 4) is shown in Theorem 4.5. □

Proof of Proposition 10.3, *part* 3. To see that $\mathrm{PCP}_{c,s}[\log, \mathrm{poly}] \subseteq \mathrm{NP}$, for every $s < c$, consider a nondeterministic machine which tries to guess an oracle which makes the verifier accept with probability at least c. The $\mathrm{NP} \subseteq \mathrm{PCP}_{c,s}[\log, 2]$ result follows from the hardness of approximating Max2SAT. Specifically, suppose that $L \leq_D^K \mathsf{Gap\text{-}2SAT}_{c,s}$. Then we can present a $\mathrm{PCP}_{c,s}[\log, 2]$ system for L as follows. On input x, the verifier in this system performs the reduction (of L to the promise problem) obtaining a 2CNF formula ϕ_x. Next it uniformly selects a clause of ϕ_x and queries the oracle for the values of the variables in this clause (accepting accordingly). Using Theorem 4.6 (part 3), $\mathrm{NP} \leq_D^K \mathsf{Gap\text{-}2SAT}_{c,s}$ for some $c > 0.9$ and $s < \frac{73}{74} \cdot c$, and $\mathrm{NP} \subseteq \mathrm{PCP}_{c,s}[\log, 2]$ follows. □

Remark 10.4. The ratio c/s has been subsequently increased to $(10/9) - \epsilon$, for any $\epsilon > 0$ (cf., [84, 57]).

Proof of Proposition 10.3, *part* 5. The result for general verifiers follows from Lemma 4.11 and the fact that MaxSAT can be approximated to within a $0.795 = 0.75 + \frac{0.18}{4}$ factor in polynomial time (cf. [84]). The (tedious) proof of the nonadaptive

case can be found in earlier versions of this paper [20]. The paper of Trevisan et al. [84] contains a stronger result which holds for all verifiers; that is, $\mathrm{PCP}_{1,0.367}[\log, 3] = \mathrm{P}$. □

The latter result (i.e., $\mathrm{PCP}_{1,0.367}[\log, 3] = \mathrm{P}$) is weaker than what can be proven for MIP proof systems (see next corollary). This contrast may provide a testing ground to separate PCP from MIP, a question raised by [21].

COROLLARY 10.5. *For $s < 1/2$, $\mathrm{MIP}_{1,s}[\mathsf{coins} = \log, \mathsf{provers} = 3] = \mathrm{P}$.*

Proof. Combining (the two parts of) Lemma 10.1 and (part 2 of) Proposition 10.3, we have $\mathrm{MIP}_{1,s}[\log, 3] \subseteq \mathrm{MIP}_{1,2s}[\log, 2] \subseteq \mathrm{PCP}_{1,2s}[\log, 2] \subseteq \mathrm{P}$. □

A general result that relates the query complexity of a probabilistically checkable proof system and the ratio between the acceptance probabilities of yes-instances and no-instances follows.

LEMMA 10.6. *For all admissible functions c, s, q, r, l such that $c/s > 2^q$,*

$$\mathrm{PCP}_{c,s}[r, q] \subseteq \mathrm{RTIME}\left(\mathrm{poly}\left(\frac{n}{c - 2^q s}\right)\right).$$

Furthermore, $\mathrm{PCP}_{c,s}[r, q] \subseteq \mathrm{PSPACE}$, and if r and q are both logarithmically bounded, then $\mathrm{PCP}_{c,s}[r, q] = \mathrm{P}$.

Proof. Let $L \in \mathrm{PCP}_{c,s}[r, q]$ and let V be a verifier demonstrating this fact. Observe that for $x \in L$, the probability that V accepts x, given access to a random oracle, is at least $\frac{c}{2^q}$. On the other hand, for $x \notin L$, the probability that V accepts x, given access to any oracle, is at most $s < \frac{c}{2^q}$. Thus, we can decide if x is in L by simulating the execution of V with access to a random oracle and estimating the acceptance probability over V's random choices and all possible oracles. In particular, we can estimate this probability up to an $\epsilon \stackrel{\mathrm{def}}{=} \frac{1}{2} \cdot (s - \frac{c}{2^q})$ additive term, with very high probability, by taking $\mathrm{poly}(1/\epsilon)$ samples. Alternatively, we can compute this probability in polynomial space. Finally, in case r and q are both logarithmically bounded, we can (exactly) compute the probability that V accepts x, given access to a random oracle. To this end we loop through all possible random pads for V and, for each pad, consider all possibilities of setting the oracle bits examined by V. Thus, for $s < \frac{c}{2^q}$, we get a deterministic polynomial-time decision procedure. □

The last assertion in the above lemma (i.e., $\mathrm{PCP}_{c,s}[\log, q] = \mathrm{P}$ for $c/s > 2^q$) cannot be strengthen by omitting the (logarithmic) bound on q since $\mathrm{NP} = \mathrm{PCP}_{1,0}[0, \mathrm{poly}]$. On the other hand, recalling the definition of $\overline{\mathrm{PCP}}$ we immediately get the following.

COROLLARY 10.7. *Let $\epsilon : \mathcal{Z}^+ \to [0, 1]$ be an admissible function strictly greater than 0. Then, for every admissible function $c : \mathcal{Z}^+ \to [0, 1]$,*

$$\overline{\mathrm{PCP}}_c[\log, 1 - \epsilon] = \mathrm{P}.$$

In particular, this holds for $c = 1$.

Proof. $L \in \overline{\mathrm{PCP}}_c[\log, 1 - \epsilon]$ implies that for some logarithmically bounded function m, we have $L \in \mathrm{PCP}_{c,2^{-m} \cdot c}[\log, (1 - \epsilon) \cdot m]$ and the corollary follows. □

PCP WITH SUPER-LOGARITHMIC RANDOMNESS. The above results are focused on pcp systems with logarithmic randomness. Proof systems with unrestricted randomness (as considered in the next proposition) may also provide some indication to the effect of very low query complexity. The results we obtain are somewhat analogous to those of Proposition 10.3. Recall that $\mathrm{PCP}_{1,1/2}[\mathrm{poly}, \mathrm{poly}]$ equals NEXPT (nondeterministic exponential time) [11]. Thus, the power of pcp systems with polynomial randomness has to be compared against NEXPT.

PROPOSITION 10.8 (general PCP systems with at most three queries).

(1) (PCP with one query is relatively very weak). *For all admissible functions* $s, c : \mathcal{Z}^+ \to [0,1]$*, so that* $c(n) - s(n)$ *is nonnegligible,*[11]

$$\mathrm{PCP}_{c,s}[\mathrm{poly}, 1] \subseteq \mathrm{AM},$$

where AM is the class of languages having one-round Arthur–Merlin proof systems (*cf.* [10]).

(2) (one-sided error pcp with two queries is relatively weak). *For all admissible functions* $s : \mathcal{Z}^+ \to [0,1]$ *strictly less than* 1, $\mathrm{PCP}_{1,s}[\mathrm{poly}, 2] \subseteq \mathrm{PSPACE}$.

(3) (two-sided error pcp with two queries is not weak). *On the other hand, there exists* $0 < s < c < 1$ *so that* $\mathrm{PCP}_{c,s}[\mathrm{poly}, 2] = \mathrm{NEXPT}$.

(4) (one-sided error pcp with three queries is not weak). $\mathrm{PCP}_{1,0.85+\epsilon}[\mathrm{poly}, 3] = \mathrm{NEXPT}$, $\forall \epsilon > 0$.

(5) (one-sided error pcp with three queries is not very strong). $\forall s < \frac{1}{8}$, $\mathrm{PCP}_{1,s}[\mathrm{poly}, 3] = \mathrm{PSPACE}$.

The first part of the proposition may be hard to improve since, as indicated in Proposition 10.9, part 6, graph nonisomorphism is in $\mathrm{PCP}_{1,1/2}[\mathrm{poly}, 1]$.

Proof of Proposition 10.8, *part* 1. We first observe that a 1-query pcp system is actually a one-round interactive proof system (cf. [53]). (The completeness and soundness bounds are as in the pcp system.) Using well-known transformations we obtain the claimed result. Specifically, we first reduce the error of the interactive proof by parallel repetition, next transform it into an Arthur–Merlin interactive proof [54], and finally transform it into an Arthur–Merlin interactive proof of perfect completeness [46]. We stress that all the transformations maintain the number of rounds up to a constant and that the constant-round Arthur–Merlin hierarchy collapses to one round [10]. □

Proof of Proposition 10.8, *parts* (3) *and* (4). For these parts we observe that the proof systems used in the corresponding parts of the proof of Proposition 10.3 do "scale up." Specifically, it is easy to see that the outer verifier used for all proof systems in this paper does scale up, yielding a canonical outer verifier of randomness complexity $O(\log(T(n))$ for any language in $\mathrm{Ntime}(T(n))$, provided $n < T(n) < 2^{\mathrm{poly}(n)}$. Furthermore, all inner verifiers used in the paper operate on constant sized oracles and so the composed verifier maintains the time and randomness complexities of the outer verifier. In particular, the verifier used for establishing Theorem 4.5 can be scaled up to yield part 4. The same holds for the verifier used for establishing part (3) of Proposition 10.3. (Note that although the exposition of the proof in Proposition 10.3 is in terms of reducing NP to Max2SAT, what actually happens is that the verifier used to establish the NP-hardness of Max2SAT (cf. section 4.2) is implemented by a verifier which makes only two queries (out of a constant number of possibilities).) □

Proof of Proposition 10.8, *part* 2. Following the strategy of the proof of the analogous part in Proposition 10.3, we obtain a polynomial-space reduction of $L \in \mathrm{PCP}_{1,s}[\mathrm{poly}, 2]$ to the set of satisfiable 2-Horn formulae (i.e., Horn formulae in which each clause has at most two literals). Namely, on input x, the reduction uses space

[11] A function $f : \mathcal{Z}^+ \to \mathcal{Z}^+$ is called nonnegligible if there exists a positive polynomial p so that $\forall n : \ f(n) > \frac{1}{p(n)}$.

poly($|x|$) and produces a Horn formula ϕ_x (of size exponential in $|x|$) so that $x \in L$ iff ϕ_x is satisfiable. Using a polylogarithmic decision procedure for satisfiability of 2-Horn formulae[12], we can decide if ϕ_x is satisfiable using poly($|x|$) space. □

Proof of Proposition 10.8, *part* 5. The result follows by the furthermore part of Lemma 10.6 (i.e., $\mathrm{PCP}_{c,s}[\mathrm{poly}, q] = \mathrm{PSPACE}$ for $c/s > 2^q$). □

10.3. Free-bit complexity. The class $\mathrm{FPCP}_{c,s}[r, f]$ is defined analogously to the class $\mathrm{PCP}_{c,s}[r, q]$ except that we consider the free-bit complexity (denoted f) instead of the query complexity (denoted q). The following proposition demonstrates the limitations of probabilistically checkable proof systems with free-bit complexity bounded by 1. We do not believe that similar limitations hold for amortized free-bit complexity.[13]

The first three items refer to proof systems with logarithmic randomness. The *second* item shows that such systems with perfect completeness and free-bit complexity 1 only exists for P (and are hence weak). In contrast, the *first* item shows the crucial role of perfect completeness in the former negative result: specifically, proof systems with two-sided error (nonperfect completeness) having free-bit complexity *zero* suffice for $\mathcal{NP}$. The *third* item asserts that the second item cannot be strengthened with respect to increasing the free-bit complexity. Proof systems with unrestricted randomness (as considered in the last three items) may also provide some indication to the effect of very low free-bit complexity. The last item can be viewed as (weak) evidence that the result in the fourth item cannot be "drastically improved" (e.g., to yield $\mathrm{FPCP}_{1,s}[\mathrm{poly}, 0] \subseteq \mathrm{BPP}$).

We make essential use of the ability to efficiently generate accepting computations, and the results may not hold otherwise.[14]

PROPOSITION 10.9 (PCP systems with low free-bit complexity). *Let* $s : \mathcal{Z}^+ \to [0, 1]$ *be an admissible function strictly smaller than* 1. *Then we have the following.*

(1) (PCP with logarithmic randomness and 0 free bit).

(1.1) *There exists* $s < 0.794$ *so that*

$$\mathrm{NP} \subseteq \mathrm{FPCP}_{\frac{1}{4},\frac{s}{4}}[\log, 0] .$$

Thus, $\mathrm{NP} = \overline{\mathrm{FPCP}}_{\frac{1}{4}}[\log, 0]$.

[12]For example, consider the following procedure. Given a 2-Horn formula, we construct a directed graph in which the vertices are the literals of the formula and there is a directed edge from literal x to literal y if the formula contains the clause $x \to y$. One can easily verify that the formula is not satisfied iff there exists a variable for which every truth assignment yields a contradiction (i.e., "forcing paths" to contradicting values; cf. [35]). Thus, a nondeterministic logspace machine can guess this variable and check that both possible truth assignments (to it) yield contradictions. The latter checking reduces to guessing the variable for which a conflicting assignment is implied and verifying the conflict via s-t directed connectivity. Since the latter task is in $\mathcal{NL}$, we are done. (Actually, 2SAT is complete for co$\mathcal{NL}$; see [60].)

[13]The conjecture was stated for systems with perfect completeness and has been subsequently proven by Håstad [56] (who proved that $\mathrm{NP} = \overline{\mathrm{FPCP}}_1[\log, \epsilon]$ for every $\epsilon > 0$). For systems with two-sided error probability, we knew that they can recognize $\mathcal{NP}$ languages using zero free bits; see below.

[14]We note, however, that the more relaxed notion of free bits may be less relevant to proving hardness of approximation results.

(1.2) *For every* $\epsilon > 0$, $\mathrm{NP} \subseteq \mathrm{FPCP}_{1-\epsilon,1-\frac{16}{15}\cdot\epsilon}[\log, 0]$.
(1.3) *For every* $\epsilon > 0$, $\mathrm{FPCP}_{1-\epsilon,1-2\cdot\epsilon}[\log, 0] \subseteq \mathrm{P}$.
(2) (limitations of PCP with logarithmic randomness and 1 free bit). $\mathrm{FPCP}_{1,s}[\log, 1] = \mathrm{P}$. *Also,* $\mathrm{FPCP}_{1,1-(1/\,\mathrm{poly})}[\mathsf{coins} = \mathrm{poly}\,;\, \mathsf{free} = 1\,;\, \mathsf{pflen} = \mathrm{poly}] \subseteq \mathrm{BPP}$.
(3) ("tightness" of item 2). *There exists* $s < 0.794$ *so that*
(3.1) $\mathrm{NP} \subseteq \mathrm{FPCP}_{1,s}[\log, 2]$;
(3.2) $\mathrm{NP} \subseteq \mathrm{FPCP}_{1,\frac{1+s}{2}}[\log, f]$ *where* $f = \log_2 3$ (i.e., $2^f = 3$);
(3.3) $\mathrm{NP} \subseteq \mathrm{FPCP}_{\frac{1}{2},\frac{s}{2}}[\log, 1]$.
(4) (general pcp with 0 free bit). $\mathrm{FPCP}_{1,s}[\mathrm{poly}, 0] \subseteq \mathrm{coNP}$.
(5) (general pcp with 1 free bit). $\mathrm{FPCP}_{1,s}[\mathrm{poly}, 1] \subseteq \mathrm{PSPACE}$.
(6) (examples for pcp with 0 free bit). *Graph nonisomorphism,* GNI, *has a PCP system with perfect completeness and soundness bound* $1/2$, *in which the verifier makes a single query and this query is free, namely,*

$$\mathtt{GNI} \in \mathrm{FPCP}_{1,1/2}[\mathsf{coins} = \mathrm{poly}\,;\, \mathsf{free} = 0\,;\, \mathsf{query} = 1].$$

The same holds for QNR (*"quadratic nonresiduosity" (cf.* [53])) *the set of integer pairs* (x, N) *so that* x *is a quadratic nonresidue modulo* N.

Proof of Proposition 10.9, *part* 3. The first claim of part 3 is justified by Theorem 5.4. Applying Proposition 11.9 to this verifier (which indeed satisfies the condition of this proposition) yields the second claim of part 3. Applying Proposition 11.8 to the same verifier (with $k = 1 < f = 2$), the third claim of part 3 follows. □

Proof of Proposition 10.9, *part* 1. Applying Proposition 11.8 (with $k = f = 2$) to the the verifier of Theorem 5.4, the first claim of part 1 follows. To prove the second claim, we apply Proposition 11.10 to the first claim and obtain $\mathrm{NP} \subseteq \mathrm{FPCP}_{1-\delta\cdot(1-0.25),1-\delta\cdot(1-0.2)}[\log, 0]$ (which holds for any δ). Substituting $\delta = (\frac{4}{3})\epsilon$, the second claim follows.

The last claim follows by the relationship between the Minimum Vertex Cover problem and the class $\mathrm{FPCP}_{c,s}[\log, 0]$; see the proof of Proposition 5.6. Specifically, consider the FGLSS reduction/graph of a proof system witnessing $L \in \mathrm{FPCP}_{1-\epsilon,1-2\epsilon}[\log, 0]$ (actually consider the complement graph where one asks about the size of the independent set). Then, for each $x \in L$ this graph has a vertex cover of density at most ϵ, whereas for $x \notin L$ this graph has no vertex cover of density 2ϵ. Using Gavril's approximation algorithm (cf. [48]), these two cases are distinguishable in polynomial time and so the third claim follows. □

Proof of Proposition 10.9, *part* 4. Let $L \in \mathrm{PCP}_{1,s}[\mathrm{poly}, 0]$ and V be a verifier demonstrating this fact. By definition, for every possible sequence of coin tosses for V, there exists at most one accepting configuration (of oracle answers to the queries made by V). Furthermore, by definition this accepting configuration (if it exists) can be generated in polynomial time from the coin sequence. Following is a nondeterministic procedure that accepts $\overline{L}$. It starts by guessing two sequences of coin tosses for V, generating the corresponding accepting configurations and checking whether they are consistent. (The input is accepted by this nondeterministic procedure iff the two coin sequences guessed yield conflicting configurations.) Clearly, if $x \in L$, then, for all possible pairs of coin sequences, accepting configurations exist and are consistent (since an oracle which *always* makes V accept x does exist). Thus, $x \in L$ is never accepted by the nondeterministic procedure. On the other hand, if all pairs of coin sequences yield accepting and mutually consistent configurations, then an oracle which

always makes V accept x emerges. Thus, for every $x \notin L$ there exists a guess which makes the nondeterministic procedure accept x. □

Proof of Proposition 10.9, *parts* 2 *and* 5. Here we consider proofs with free-bit complexity 1. Thus, for each possible sequence of coin tosses, there exist at most two accepting configurations (which again can be efficiently found given the coin sequence). We refer to these two possible accepting configurations as the 1-configuration and the 2-configuration of the coin sequence. In case a specific coin sequence has less than two accepting configurations, we introduce dummy configurations so that now each coin sequence has two associated configurations. Given an input x to such a pcp system, we consider the following 2CNF formula representing all possible computations of the verifier with a generic oracle. For each possible sequence of coin tosses, $\overline{c}$, we introduce a pair of boolean variables, $\pi^1_{\overline{c}}$ and $\pi^2_{\overline{c}}$, representing which of the two associated configurations is encountered (e.g., $\pi^1_{\overline{c}} = T$ means that the 1-configuration is encountered). To enforce that a single accepting configuration is encountered we introduce the clauses $(\pi^1_{\overline{c}} \vee \pi^2_{\overline{c}})$ and $((\neg\pi^1_{\overline{c}}) \vee (\neg\pi^2_{\overline{c}}))$. In addition, in case the σ-configuration of $\overline{c}$ is not accepting (but rather a dummy configuration) we introduce the clause $(\neg\pi^\sigma_{\overline{c}})$ thus "disallowing" a computation in which it is encountered. Finally, for each pair of coin sequences we introduce clauses disallowing inconsistencies. Namely, suppose that the σ-configuration of $\overline{c}$ is inconsistent with the τ-configuration of $\overline{c}'$, then we introduce the clause $((\neg\pi^\sigma_{\overline{c}}) \vee (\neg\pi^\tau_{\overline{c}'}))$, which is logically equivalent to $\neg(\pi^\sigma_{\overline{c}} \wedge \pi^\tau_{\overline{c}'})$. The resulting 2CNF formula, ϕ_x, is satisfiable if and only if there exists an oracle which causes V to accept x with probability 1. Thus, given x, we need to test if ϕ_x is satisfiable. We consider two cases.

(1) In case V uses logarithmically many coins, the 2CNF formula ϕ_x can be generated from x in polynomial time. Using a polynomial-time decision procedure for satisfiability of 2CNF formulae, we conclude that $\mathrm{FPCP}_{1,s}[\log, 1] = \mathrm{P}$. Furthermore, using Proposition 11.2, we can randomly reduce

$$\mathrm{FPCP}_{1,1-(1/\,\mathrm{poly})}[\mathrm{poly}, \mathsf{free} = 1, \mathsf{pflen} = \mathrm{poly}]$$

to

$$\mathrm{FPCP}_{1,1-(1/\,\mathrm{poly})}[\log, \mathsf{free} = 1]\ ,$$

and

$$\mathrm{FPCP}_{1,1-(1/\,\mathrm{poly})}[\mathrm{poly}, \mathsf{free} = 1, \mathsf{pflen} = \mathrm{poly}] \subseteq \mathrm{BPP}$$

follows. This establishes part 2.

(2) In general (V may make polynomially many coin tosses), the 2CNF formula ϕ_x may have exponential (in $|x|$) length. Yet it can be generated from x in polynomial space. Using a polylogarithmic-space decision procedure for satisfiability of 2CNF formulae,[15] we can decide if ϕ_x is satisfiable using $\mathrm{poly}(|x|)$ space. Part 5 (i.e., $\mathrm{FPCP}_{1,s}[\mathrm{poly}, 1] \subseteq \mathrm{PSPACE}$) follows. □

Proof of Proposition 10.9, *part* 6. We merely note that the interactive proof presented in [52] for graph nonisomorphism[16] constitutes a 1-query pcp system with

[15] For example, note that 2CNF formulae can be written in Horn form and use the procedure described in the proof of Proposition 10.8, part 2.

[16] On input a pair of graphs, G_0 and G_1, the verifier uniformly selects $i \in \{0, 1\}$ and generates a random isomorphic copy of G_i, denoted H. This graph H is the single query made by the verifier, which accepts if and only if the answer equals i.

perfect completeness and soundness bound 1/2. Furthermore, the query made by the verifier has a unique, acceptable answer and thus the free-bit complexity of this system is zero. The same holds for the interactive proof presented in [53] for quadratic nonresiduosity QNR, which is actually the inspiration for the proof in [52]. □

10.4. Query complexity versus free-bit complexity. The following proposition quantifies the intuition that not all queries are "undetermined" (i.e., that the free-bit complexity is lower than the query complexity). Furthermore, as a corollary we obtain that the amortized (average) free-bit complexity is at least one unit less than the amortized query complexity.

PROPOSITION 10.10. *For admissible functions c, s, r, q such that $r(n), q(n) = O(\log n)$,*

$$\mathrm{PCP}_{c,s}[r,q] \subseteq \mathrm{PCP}_{c,s}[\,\mathsf{coins} = r\,;\,\mathsf{free}_{\mathrm{av}} = q - \log_2(1/s)\,]. \tag{23}$$

Furthermore, for every admissible function t, $\mathrm{PCP}_{c,s}[r,q] \subseteq \mathrm{FPCP}_{c,(2^t+1)\cdot s}[r, q-t]$.

Proof. Let $L \in \mathrm{PCP}_{c,s}[r,q]$ and let V be the verifier demonstrating this. Fix an input $x \in \Sigma^n$, and let $r = r(n), q = q(n), s = s(n)$. For a random string $R \in \{0,1\}^r$, let F_R^x denote the number of accepting patterns of V, i.e., $F_R^x = |\mathsf{pattern}_V(x;R)|$. We first claim that if $\mathbf{E}_R[F_R^x] > 2^q \cdot s$, then $x \in L$. This is true since a random oracle π is accepted with probability at least $\mathbf{E}_R[F_R^x \cdot 2^{-q}]$, and so if the claim were not to hold we would have reached contradiction to the soundness condition (i.e., $x \notin L$ is accepted with probability strictly larger than s).

We now construct a verifier, denoted V', witnessing $L \in \mathrm{FPCP}^{\mathrm{av}}_{c,s}[r, q-\log_2(1/s)]$. On input x, the verifier first computes $\mathbf{E}_R[F_R^x]$ (by scanning all possible R's and generating all accepting patterns for each of them). If $\mathbf{E}_R[F_R^x]) > 2^q \cdot s$, then V' accepts x (without querying the oracle). Otherwise (i.e., if $\mathbf{E}_R[F_R^x]) \leq 2^q \cdot s$), then V' simulates V and accepts if V accepts. It follows that the average free-bit complexity of V' on input x equals the corresponding quantify for V, provided the latter is at most $q - \log_2(1/s)$, and equals zero otherwise. The first part of the proposition follows.

To establish the second part, for some $t = t(n)$, we construct a verifier V'' which, on input x, proceeds as follows. First, V'' computes $q \stackrel{\mathrm{def}}{=} \mathbf{E}_R[F_R^x]$ and accepts if $q > s2^q$ (just as V'). In case $q \leq s2^q$, the new verifier proceeds differently: it randomly selects R as V does and computes F_R^x. If $F_R^x > 2^{q-t}$, then V'' accepts and otherwise it invokes V on input x and coins R. Clearly, this guarantees that the free-bit complexity of V'' is at most $q-t$. To analyze the soundness of V'', note that when $\mathbf{E}_R[F_R^x] \leq s2^q$, it follows that $\Pr_R[F_R^x > 2^{q-t}] \leq 2^t \cdot s$ (Markov inequality). Thus, the soundness error of V'' is at most $s + 2^t s$ and the second part follows. □

By computing the amortized average free-bit complexity of the class of languages in the right-hand side of Eq. (23) above, we obtain the following consequence.

COROLLARY 10.11. *For admissible functions c, r, q with $r(n), q(n) = O(\log n)$,*

$$\overline{\mathrm{PCP}}_c[\,r,\,q\,] \subseteq \overline{\mathrm{FPCP}}^{\mathrm{av}}_c[\,r,\,q-1\,],$$

where $\overline{\mathrm{FPCP}}^{\mathrm{av}}_{\cdot}[\,\cdot,\,f\,]$ denotes a class analogous to $\overline{\mathrm{FPCP}}_{\cdot}[\,\cdot,\,f\,]$ in which average free-bit complexity is measured instead of (worst-case) free-bit complexity.

Proof. For some function m, we have

$$\overline{\mathrm{PCP}}_c[\,r,\,q\,] \subseteq \mathrm{PCP}_{c,c\cdot 2^{-m}}[r, qm] \subseteq \mathrm{FPCP}^{\mathrm{av}}_{c,c\cdot 2^{-m}}[r, qm-m] \subseteq \overline{\mathrm{FPCP}}^{\mathrm{av}}_c[\,r,\,q-1\,],$$

where the second inclusion is due to Eq. (23). □

The above corollary clinches the argument that the amortized query complexity is incapable of capturing the approximability of the clique function. Suppose that, for some g (e.g., $g = \frac{3}{2}$), MaxClique is NP-hard to approximate to within an $N^{1/(1+g)}$ factor, but it can be approximated to within an $N^{1/(1+g-\delta)}$ factor in polynomial time, for every $\delta > 0$ (actually, it suffices to postulate that MaxClique can be approximated to within an $N^{1/g}$ factor in polynomial time). Furthermore, supposed that the hardness result is demonstrated by showing that $\mathrm{NP} \subseteq \overline{\mathrm{PCP}}[\log, g-\epsilon]$, for every $\epsilon > 0$. Then, using the above corollary, we get $\mathrm{NP} \subseteq \overline{\mathrm{FPCP}}^{\mathrm{av}}[\log, g-1-\epsilon]$, for every $\epsilon > 0$, and an NP-hardness result of clique approximation[17] up to an $N^{1/(1+(g-1-\epsilon)+\epsilon)} = N^{1/g}$ follows, in contradiction to our hypothesis that such approximations could be achieved in polynomial time. To summarize, attempts to establish the factor $N^{1/(g+1)}$, for which it is NP-hard to approximate MaxClique via amortized query complexity, will always fall at least one unit away from the truth, whereas amortized free-bit complexity will yield the right answer.

11. Transformations of FPCP systems. We present several useful transformations which can be applied to pcp systems. These fall into two main categories:

(1) Transformations which amplify the (completeness versus soundness) gap of the proof system, while preserving (or almost preserving) its amortized free-bit complexity.
(2) Transformations which move the gap location (or, equivalently, the completeness parameter). The gap itself is almost preserved but moving it changes the free-bit complexity (and thus the amortized free-bit complexity is not preserved). Specifically, moving the gap "up" requires increasing the free-bit complexity, whereas moving the gap "down" allows us to decrease the free-bit complexity.

Most of these transformations are analogous to transformations which can be applied to graphs with respect to the MaxClique approximation problem. In view of the relation between FPCP and the clique promise problem (shown in section 8), this analogy is hardly surprising.

In this section, we use a more extensive FPCP notation which refers to promise problems (rather than to languages) and introduce an additional parameter—the proof length. Specifically, $\mathrm{FPCP}_{c,s}[r, f, l]$ refers to randomness complexity r, free-bit complexity f, and proof-length l.

11.1. Gap amplifications maintaining amortized free-bit complexity. We start by stating the simple fact that the ratio between the completeness and soundness bounds (also referred to as gap) is amplified (i.e., raised to the power k) when one repeats the pcp system (k times). Note, however, that if the original system is not perfectly complete, then the completeness bound in the resulting system gets decreased.

PROPOSITION 11.1 (simple gap amplification). *For all* $c, s : \mathcal{Z}^+ \to [0,1]$ *and* $k : \mathcal{Z}^+ \to \mathcal{Z}^+$,

$$\mathrm{FPCP}_{c,s}[r, f, l] \subseteq \mathrm{FPCP}_{c^k,s^k}[kr, kf, l].$$

Proof. Let $(Y, N) \in \mathrm{FPCP}_{c,s}[r, f, l]$ and let V be a verifier witnessing this with query complexity $q : \mathcal{Z}^+ \to \mathcal{Z}^+$. Given $k : \mathcal{Z}^+ \to \mathcal{Z}^+$, we define a verifier $V^{(k)}$ as

[17] Here we use the observation that the FGLSS reduction works also for amortized *average* free-bit complexity.

follows: on input $x \in \{0,1\}^n$, let $r = r(n), k = k(n), f = f(n), l = l(n)$, and $q = q(n)$.

- $V^{(k)}$ picks k random strings $\overline{c}^{(1)}, \ldots, \overline{c}^{(k)}$ uniformly and independently in $\{0,1\}^r$.
- For $i = 1$ to k, verifier $V^{(k)}$ simulates the actions of V on input x and random string $\overline{c}^{(i)}$. Verifier $V^{(k)}$ accepts if V accepts on each of these k instances.

Clearly, $V^{(k)}$ tosses kr coins and examines the l-bit long oracle in at most kq bits, where at most kf of these are free. For every x, if the probability that V accepts x, given access to oracle π, is p, then the probability that $V^{(k)}$ accepts x, given access to π, is exactly p^k. Thus, $(Y, N) \in \mathrm{FPCP}_{c^k, s^k}[kr, kf, l]$, and oracles can be transformed (by identity) from one pcp system to the other. □

Next, we show that in some sense the randomness complexity of a proof system need not be higher than logarithmic in the length of the proofs/oracles employed. Specifically, we show how to randomly reduce languages proven by the first kind of systems into languages proven by the second kind. Thus, whenever one is interested in the computational complexity of languages proven via pcp systems, one may assume that the system is of the second type. Recall that $\leq_R^K$ denotes a randomized Karp reduction.

PROPOSITION 11.2 (reducing randomness). *There exists a constant $\gamma > 0$ so that*

(1) (for perfect completeness): *for every two admissible functions $s, \epsilon : \mathcal{Z}^+ \to [0,1]$,*

$$\mathrm{FPCP}_{1,s}[r, f, l] \leq_R^K \mathrm{FPCP}_{1,s'}[r', f, l],$$

where $s' = (1+\epsilon) \cdot s$ and $r' = \gamma + \log_2(l/\epsilon^2 s)$.

(2) (for two-sided error): *for every four admissible functions $c, s, \epsilon_1, \epsilon_2 : \mathcal{Z}^+ \to [0,1]$,*

$$\mathrm{FPCP}_{c,s}[r, f, l] \leq_R^K \mathrm{FPCP}_{c',s'}[r', f, l],$$

where $c' = 1 - (1+\epsilon_1) \cdot (1-c) \geq c - \epsilon_1$, $s' = (1+\epsilon_2) \cdot s$ and $r' = \gamma + \max\{-\log_2(\epsilon_1^2(1-c)),\ \log_2(l) - \log_2(\epsilon_2^2 s)\}$.

Proof. The proof is reminiscent of Adleman's proof that $\mathcal{RP} \subseteq \mathrm{P/poly}$ [1]. Suppose we are given a pcp system for which we want to reduce the randomness complexity. The idea is that it suffices to choose the random pad for the verifier out of a relatively small set of possibilities (instead of from all 2^r possibilities). Furthermore, most small sets (i.e., sets of size linear in l) are good for this purpose. This suggest randomly mapping an input x for the original pcp system into an input (x, R) for the new system, where R is a random set of $m = O(l)$ possible random pads for the original system. The new verifier will select a random pad uniformly in R, thus using only $\log_2 |R|$ random coins, and run the original verifier using this random pad. Details follow.

We start with the simpler case stated in part 1. Let $(Y, N) \in \mathrm{FPCP}_{1,s}[r, f, l]$ and let V be a verifier demonstrating this fact. The random reduction maps $x \in \{0,1\}^n$ to (x, R), where R is a uniformly chosen m multisubset of $\{0,1\}^r$ for $l \stackrel{\mathrm{def}}{=} l(n)$, $r \stackrel{\mathrm{def}}{=} r(n)$, $s \stackrel{\mathrm{def}}{=} s(n)$, $\epsilon \stackrel{\mathrm{def}}{=} \epsilon(n)$, and $m \stackrel{\mathrm{def}}{=} \frac{\gamma l}{\epsilon^2 s}$. (The constant γ is chosen to make the Chernoff bound, used below, hold.) On input (x, R), the new verifier V' uniformly selects $\overline{c} \in R$ and invokes V with input x and random pad $\overline{c}$. Clearly, the complexities of V' are as claimed above. Also, assuming that V always accepts x, when given access to an oracle π, then, for every possible pair (x, R) to which x is mapped, V' always accepts

(x, R) when given access to the oracle π. It remains to upper bound, for each $x \notin L$ and most R's, the probability that V' accepts (x, R) when given access to an arbitrary oracle.

Fixing any $x \notin L$ and any oracle π, we bound the probability that V', give access to π, accepts (x, R) for most R's. A set R is called *bad* for x with respect to π if for more than an s' fraction of the $\overline{c} \in R$ the verifier V accepts x when given access to π and random pad $\overline{c}$. Let $R = (\overline{r}^{(1)}, \ldots, \overline{r}^{(m)})$ be a uniformly selected multiset. For every $i \in [m]$ (a possible random choice of V'), we define a 0-1 random variable ζ_i so that it is 1 iff V on random pad $\overline{r}^{(i)}$ and access to oracle π accepts the input x. Clearly, the ζ_i's are mutually independent and each equals 1 with probability $\delta \leq s$. Using a multiplicative Chernoff bound (cf. [75, Theorem 4.3]), the probability that a random R is bad (for x w.r.t. π) is bounded by

$$\Pr\left[\sum_{i=1}^{m} \zeta_i \geq (1+\epsilon) \cdot ms\right] < 2^{-\Omega(\epsilon^2 \cdot ms)}.$$

Thus, by the choice of m, the probability that a random R is bad for x, with respect to any fixed oracle, is smaller than $\frac{1}{4} \cdot 2^{-l}$. Since there are only 2^l relevant oracles, the first part of the proposition follows.

For the second part of the proposition, we repeat the same argument, except that now we need to take care of the completeness bound in the resulting pcp system. This is done similarly to the way we deal with the soundness bound, except that we do not need to consider all possible oracles; it suffices to consider the best oracle for each $x \in Y$. When applying the multiplicative Chernoff bound it is important to note that, since we are interested in the rejection event, the relevant expectation is $m \cdot (1-c)$ (and not $m \cdot c$). Thus, as long as $m \geq \frac{2\gamma}{\epsilon_1^2(1-c)}$, at least 3/4 of the possible sets R cause V' to accept $x \in Y$ with probability at least $1 - (1+\epsilon_1) \cdot (1-c) = c - (1-c)\epsilon_1$. The second part of the proposition follows. □

Combining Propositions 11.1 and 11.2, we obtain a randomized reduction of pcp systems which yields the effect of Proposition 11.1 at much lower (and in fact minimal) cost in the randomness complexity of the resulting pcp system. This reduction is analogous to the well-known transformation of Berman and Schnitger [25]. The reduction (in either forms) plays a central role in deriving clique approximation results via the FGLSS method: applying the FGLSS reduction to proof systems obtained via the second item (below), one derives graphs of size $N \stackrel{\text{def}}{=} 2^{(1+\epsilon+f)\cdot t}$ with clique gap 2^t (which can be rewritten as $N^{1/(1+f+\epsilon)}$). For sake of simplicity, we only state the case of perfect completeness.

COROLLARY 11.3 (probabilistic gap amplification at minimal randomness cost).

(1) (Combining the two propositions.) *For every admissible $k : \mathcal{Z}^+ \to \mathcal{Z}^+$,*

$$\begin{aligned}&\mathrm{PCP}_{1,1/2}[\,\mathsf{coins} = r\,;\,\mathsf{query} = q\,;\,\mathsf{free} = f\,;\,\mathsf{pflen} = l\,]\\ &\leq_R^K \mathrm{FPCP}_{1,2^{-k+1}}[r + \log_2 q + O(1) + k, kf, l]\,.\end{aligned}$$

(2) (Using amortized free-bit complexity.) *For every $\epsilon > 0$, there exists a constant c so that*

$$\overline{\mathrm{FPCP}}[\log,\, f, l\,] \leq_R^K \mathrm{FPCP}_{1,2^{-t}}[(1+\epsilon)\cdot t, f \cdot t, l],$$

where $t(n) = c \log_2 n$.

Proof. Suppose that $(Y,N) \in \mathrm{FPCP}_{1,1/2}[r,f,l]$. Clearly, $l \le 2^r \cdot q$, where $q(n) = \mathrm{poly}(n)$ is the query complexity of the verifier. Then, applying Proposition 11.1, we get $(Y,N) \in \mathrm{FPCP}_{1,1/2^k}[kr, kf, 2^r \cdot q]$. Applying part 1 of Proposition 11.2, we obtain $(Y,N) \le_R^K \mathrm{FPCP}_{1,\frac{1}{2^{k-1}}}[r', kf]$, where $r' = O(1) + \log_2(2^r q/2^{-k}) = O(1) + r + k + \log_2 q$. The first part of the corollary follows.

Suppose now that a language L has a proof system as in the hypothesis of the second part. Then, there exists a logarithmically bounded function m so that $L \in \mathrm{FPCP}_{1,1/2^m}[r, mf, l]$, where $r(n) \le \alpha \cdot \log_2 n$ and $l(n) \le n^\beta$ for some constants α and β. Invoking a similar argument (to the above), we get $L \le_R^K \mathrm{FPCP}_{1,\frac{1}{2^{km-1}}}[r', k \cdot mf]$, where $r'(n) = O(1) + km + (\alpha+\beta) \cdot \log_2 n$. Now, setting $k(n)$ so that $k(n) \cdot m(n) \ge \frac{\alpha+\beta}{\epsilon} \cdot \log_2 n$, the corollary follows. □

An alternative gap amplification procedure which does not employ randomized reductions is presented below. This transformation increases the randomness complexity of the pcp system more than the randomized reduction presented above (i.e., $r' \approx O(r) + 2k$ rather than $r' \approx r + k$ as in item 1 of Corollary 11.3). The transformation is used to obtain in-approximability results under the assumption $\mathrm{P} \ne \mathrm{NP}$ (rather than under $\mathrm{NP} \not\subseteq \mathrm{BPP}$). Again, we only state the case of perfect completeness.

PROPOSITION 11.4 (deterministic gap amplification at low randomness cost). *For every $\epsilon, s > 0$ and every admissible function $k : \mathcal{Z}^+ \to \mathcal{Z}^+$,*

$$\mathrm{FPCP}_{1,s}[r,f,l] \subseteq \mathrm{FPCP}_{1,s^k}[O(r) + (2+\epsilon) \cdot k \cdot \log(1/s), (1+\epsilon) \cdot kf, l].$$

Actually, the constant in the O notation is $\min\{1, \frac{2+(4/\epsilon)}{\log_2(1/s)}\}$.

We use random walks on expander graphs for error reduction (cf. [2, 30]). The value of the constant multiplier of $k \log(1/s)$ in the randomness complexity of the resulting pcp system depends on the expander graph used. Specifically, using a degree d expander graph with second eigenvalue λ yields a factor of $\frac{\log_2 d}{1+\log_2 \lambda}$. Thus, it is essential to use Ramanujan graphs [70] in order to obtain the claimed constant of $2+\epsilon$.

Proof of Proposition 11.4. For simplicity assume $s = 1/2$. The idea is to use a "pseudorandom" sequence generated by a random walk on an expander graph in order to get error reduction at moderate randomness cost. Specifically, we will use a Ramanujan expander graph of constant degree d and second eigenvalue $\lambda \approx 2\sqrt{d}$ (cf. [70]). The constant d will be determined so that $d > 2^{4+\frac{8}{\epsilon}}$ (and $d < 2^{6+\frac{8}{\epsilon}}$). It is well known that a random walk of length t in an expander avoids a set of density ρ with probability at most $(\rho + \frac{\lambda}{d})^t$ (cf. [2, 61]). Thus, as a preparation step we reduce the error probability of the pcp system to

$$p \stackrel{\mathrm{def}}{=} \frac{\lambda}{d} = \frac{2}{\sqrt{d}}. \tag{24}$$

This is done using the trivial reduction of Proposition 11.1. We derive a proof system with error probability p, randomness complexity

$$r' \stackrel{\mathrm{def}}{=} r \cdot \log_2(1/p) = r \cdot \log_2(\sqrt{d}/2) = O(r), \tag{25}$$

and free-bit complexity

$$f' \stackrel{\mathrm{def}}{=} f \cdot \log_2(1/p) = f \cdot \log_2(\sqrt{d}/2). \tag{26}$$

(In case we start with soundness error s, where $s > p$, the multiplier will be $\log_{1/s}(1/p)$ instead of $\log_2(1/p)$.) Now we are ready to apply the expander walk technique. Using an expander walk of length t, we transform the proof system into one in which the randomness complexity is $r' + (t-1) \cdot \log_2 d$, the free-bit complexity is $tf' = tf \cdot \log_2(\sqrt{d}/2)$, and the error probability is at most $(2p)^t = (4/\sqrt{d})^t = 2^{-k}$, where $k \stackrel{\text{def}}{=} t \cdot \log_2(\sqrt{d}/4)$. Using $\log_2 d > \frac{8}{\epsilon} + 4$, we can bound the randomness complexity by

$$\begin{aligned} r' + t \log_2 d &= r' + \frac{\log_2 d}{\frac{1}{2} \cdot (\log_2 d) - 2} \cdot k \\ &< r' + (2+\epsilon) \cdot k. \end{aligned}$$

and the free-bit complexity by

$$\begin{aligned} tf \cdot \log_2(\sqrt{d}/2) &= \frac{\frac{1}{2} \cdot (\log_2 d) - 1}{\frac{1}{2} \cdot (\log_2 d) - 2} \cdot kf \\ &< (1+\epsilon) \cdot kf. \end{aligned}$$

The proposition follows. □

Using Proposition 11.4, we obtain the following corollary which is used in deriving clique in-approximability results under the P $\neq$ NP assumption, via the FGLSS method: applying the FGLSS reduction to proof systems obtained via this corollary, one derives graphs of size $N \stackrel{\text{def}}{=} 2^{(2+\epsilon+f)\cdot t}$ with clique gap 2^t (which can be rewritten as $N^{1/(2+f+\epsilon)}$).

COROLLARY 11.5. *For every $\epsilon > 0$, there exists a constant c so that*

$$\overline{\text{FPCP}}[\log,\ f, l\,] \subseteq \text{FPCP}_{1,2^{-t}}[(2+\epsilon)\cdot t, (1+\epsilon)\cdot f \cdot t, l],$$

where $t(n) = c \log_2 n$.

11.2. Trading off gap location and free-bit complexity. The following transformation is analogous to the randomized layering procedure for the clique promise problem (i.e., Proposition 8.6). The transformation increases the acceptance probability bounds at the expense of increasing the free-bit complexity.

PROPOSITION 11.6 (increasing acceptance probabilities and free-bit complexity).

(1) (Using a randomized reduction which preserves the randomness of the proof system.) *For all admissible functions $c, s : \mathcal{Z}^+ \to [0,1]$, and $r, f, m : \mathcal{Z}^+ \to \mathcal{Z}^+$,*

$$\text{FPCP}_{c,s}[r, f] \leq_R^K \text{FPCP}_{c',s'}[r, f + \log_2 m],$$

where $c' = 1 - 4(1-c)^m$ and $s' = m \cdot s$. In case $c' > 1 - 2^{-r}$, we then have $c' = 1$.

(2) (Inclusion which moderately increases the randomness of the proof system.) *For all admissible functions $c, s : \mathcal{Z}^+ \to [0,1]$, and $r, f, m : \mathcal{Z}^+ \to \mathcal{Z}^+$,*

$$\text{FPCP}_{c,s}[r, f] \subseteq \text{FPCP}_{c',s'}[r', f + \log_2 m],$$

- *where if* $m \leq 1/c$, *then* $r' = 2 \cdot \max\{r, \log m\}$, $c' = \frac{m}{2} \cdot c$ *and* $s' = m \cdot s$;
- *and otherwise* (*i.e., for* $m > 1/c$), $r' = O(\max\{r, \log m\} + mc)$, $c' = 1 - 2^{-\Theta(mc)}$ *and* $s' = m \cdot s$.

Proof. Suppose we are given a pcp system for which we want to increase the acceptance probability bound in the completeness condition. The idea is to allow the new verifier to select m random pads for the original verifier and query the oracle as to which pad to use. A straightforward implementation of this idea will increase the randomness complexity of the verifier by a factor of m. Instead, we use two alternative implementations which yield the two parts of the proposition. In both implementations the free-bit complexity increases by $\log_2 m$ and the soundness bound increases by a factor of m.

The first implementation employs a technique introduced by Lautemann (in the context of $\mathcal{BPP}$) [68]. Using a randomized reduction, we supply the new verifier with a sequence of m possible "shifts" that it may effect. The new verifier selects one random pad for the original verifier and generates m shifts of this pad. Now, the new verifier queries the oracle as to which of these shifts it should use as a random pad for the original verifier. Details follow.

We first present a random reduction mapping $x \in \{0,1\}^n$ to (x, S), where S is a uniformly chosen m multisubset of $\{0,1\}^r$, for $r \stackrel{\text{def}}{=} r(n)$. On input (x, S), the new verifier V' uniformly selects $\overline{c} \in \{0,1\}^r$ and queries the oracle on $(x, \overline{c})$ receiving an answer $i \in [m]$. Intuitively, V' asks which shift of the random pad to use. Finally, V' invokes V with input x and random pad $\overline{c} \oplus \overline{s}_i$, where $\overline{s}_i$ is the ith string in S. Clearly, the complexities of V' are as claimed above. Also, assuming that V accepts x with probability δ, we get that, for every S, verifier V' accepts (x, S) with probability at most $m \cdot \delta$. On the other hand suppose that, when given access to oracle π, verifier V accepts x with probability δ. It follows that there exists a set R of $\delta 2^r$ random pads for V so that if V uses any $\overline{c} \in R$ (and queries oracle π), then it accepts x. Fixing any $\overline{c} \in \{0,1\}^r$, we ask what is the probability, for a uniformly chosen $S = \{\overline{s}_i : i \leq m\}$, that there exists an $i \in [m]$ so that $\overline{c} \oplus \overline{s}_i \in R$. Clearly, the answer is $1 - (1-\delta)^m$. Thus, for uniformly chosen $S \in (\{0,1\}^r)^m$ and $\overline{c} \in \{0,1\}^r$,

$$\Pr[\, \exists i \in [m] \ \text{ s.t. } \ \overline{c} \oplus \overline{s}_i \in R \,] = 1 - (1-\delta)^m.$$

By Markov inequality, with probability at least $3/4$, a uniformly chosen $S = \{\overline{s}_i\}$ has the property that for at least $1 - 4 \cdot (1-\delta)^m$ of the $\overline{c}$'s (in $\{0,1\}^r$) there exists an $i \in [m]$ so that $\overline{c} \oplus \overline{s}_i \in R$. Part 1 of the proposition follows.

To prove part 2 of the proposition, we use an alternative implementation of the above idea, which consists of letting the new verifier V' generate a "pseudorandom" sequence of possible random pads by itself. V' will then query the oracle as to which random pad to use, in the simulation of V, and complete its computation by invoking V with the specified random pad. To generate the "pseudorandom" sequence we use the sampling procedure of [18]. Specifically, for $m \leq 1/c$ we merely generate a pairwise independent sequence of uniformly distributed strings in $\{0,1\}^r$, which can be done using randomness $\max\{2r, 2\log_2 m\}$. Otherwise (i.e., for $m > 1/c$), we use the construction of [18] to generate $\Theta(cm)$ such related sequences, where the sequences are related via a random walk on a constant degree expander. Part 2 follows. □

The following corollary exemplifies the usage of the above proposition. In case $c(n) = n^{-\alpha}$ and $r(n) = O(\log n)$, we obtain perfect completeness while preserving the gap (up to a logarithmic factor) and increasing the free-bit complexity by

a $\log_2 1/c$ additive term. Thus, the corollary provides an alternative way of deriving the reverse-FGLSS transformation (say, Proposition 8.7) from the simple clique verifier of Theorem 8.2. Specifically, one may apply the following corollary to the simple clique verifier of Theorem 8.2, instead of combining the layered-graph verifier (of Theorem 8.3) and the graph-layering process of Proposition 8.6.

COROLLARY 11.7. *For all admissible $r, f : \mathcal{Z}^+ \to \mathcal{Z}^+$, so that $\forall n : \ r(n) \geq 2$,*

$$\mathrm{FPCP}_{c,s}[r,f] \ \leq_R^K \ \mathrm{FPCP}_{1,r\cdot\frac{s}{c}}[r, f + \log_2 r + \log_2(1/c)].$$

(Compare to item 1 of Proposition 8.7.)

We conclude this subsection with another transformation which is reminiscent of an assertion made in section 8. The following transformation has an opposite effect than the previous one, reducing the free-bit complexity at the expense of lowering the bounds on acceptance probability.

PROPOSITION 11.8 (decreasing acceptance probabilities and free-bit complexity). *For all admissible functions $c, s : \mathcal{Z}^+ \to [0,1]$, and $r, f, k : \mathcal{Z}^+ \to \mathcal{Z}^+$ so that $k \leq f$, if $L \in \mathrm{FPCP}_{c,s}[r,f]$, then $L \in \mathrm{FPCP}_{\frac{c}{2^k},\frac{s}{2^k}}[r+k, f-k]$. Furthermore, in case each random pad in the original pcp system has at least 2^k accepting configurations, the average free-bit complexity of the resulting system is $f_{\mathrm{av}} - k$, where f_{av} is the average free-bit complexity of the original system.*

Proof. Let V be a verifier satisfying the condition of the proposition. We construct a new verifier V' that on input $x \in \{0,1\}^n$, setting $r = r(n)$, $k = k(n)$, and $f = f(n)$, acts as follows. Verifier V' uniformly selects a random-pad $\overline{c} \in \{0,1\}^r$ for V and generates all possible accepting configurations with respect to $V(x)$ and random-pad $\overline{c}$. In case there are less than 2^k accepting configurations we add *dummy configurations* to reach the 2^k count. We now partition the set of resulting configurations (which are accepting and possibly also dummy) into 2^k parts of about the same size (i.e., some parts may have one configuration more than others). Actually, if we only care about average free-bit complexity, then any partition of the accepting configurations into 2^k nonempty parts will do. The new verifier, V', uniformly selects $i \in [2^k]$, thus specifying one of these parts, denoted A_i. Next, V' invokes V with random-pad $\overline{c}$ and accepts if and only if the oracle's answers form an accepting configuration which is in A_i (i.e., resides in the selected portion of the accepting configurations). (We stress that in case $\overline{c}$ has less than 2^k accepting configurations and the selected A_i does not contain any accepting configuration, then V' rejects on coins $(i, \overline{c})$.) Clearly, the randomness complexity of the new verifier is $r + k$.

To analyze the other parameters of V', we fix any $x \in \{0,1\}^n$. For sake of simplicity, we first assume that the number of accepting configurations of V for any random pad is a power of 2. Then the number of accepting configurations of V' for any random-pad $(\overline{c}, i) \in \{0,1\}^r \times [2^k]$ is 2^{m-k}, where 2^m is the number of accepting configurations of V on random-pad $\overline{c}$. Thus, the free-bit complexity of V' is $f - k$. Finally, we relate the acceptance probability of V' to that of V. This is done by reformulating the execution of V' with oracle π as consisting of two steps. First V' invokes V with access to π. If V reaches a rejecting configuration, then V' rejects as well; otherwise (i.e., when V reaches an accepting configuration), V accepts with probability 2^{-k} (corresponding to uniformly selecting $i \in [2^k]$). It follows that on input x and access to oracle π, the verifier V' accepts with probability $\frac{\delta}{2^k}$, where δ denotes the probability that V accepts input x when given access to oracle π.

In general, our simplifying assumption that the number of accepting configurations of V is a power of 2 may not hold and the analysis becomes slightly more cumbersome. Firstly, the number of accepting configurations of V' for a random-pad $(\overline{c}, i)$ is either $\lceil M/2^k \rceil$ or $\lfloor M/2^k \rfloor$, where M is the number of accepting configurations of V on random-pad $\overline{c}$. Thus, in the worst case the number of accepting configurations for V' (on random-pad $(\overline{c}, i)$) is $\lceil M/2^k \rceil$, and it follows that the free-bit complexity of V' is $\log_2 \lceil 2^f/2^k \rceil = f - k$. Furthermore, the expected number of accepting configurations (for a fixed $\overline{c}$ and uniformly chosen $i \in [2^k]$) is exactly $M/2^k$ (even if $M < 2^k$). Thus, if the extra condition holds, then the free-bit complexity of V' equals $f_{\text{av}} - k$. Finally, observe that the argument regarding the acceptance probabilities remains unchanged (and actually it does not depend on the partition of the accepting configurations into 2^k nonempty parts). The proposition follows. □

11.3. Other effects on acceptance probabilities and free-bit complexity. Following is an alternative transformation which reduces the free-bit complexity. However, unlike Proposition 11.8, the following does not decrease the acceptance parameters. Furthermore, the transformation increases the soundness parameter and so does *not* preserve the gap (between the completeness and soundness parameters).

PROPOSITION 11.9 (decreasing free-bit complexity without decreasing acceptance probabilities). *Let $c, s : \mathcal{Z}^+ \to [0,1]$ be admissible functions and $r, f, k : \mathcal{Z}^+ \to \mathcal{Z}^+$. Suppose $L \in \text{FPCP}_{c,s}[r, f]$ with a verifier for which the first k oracle-answers for each random pad allow at most 2^{f-k} accepting configurations. Then $L \in \text{FPCP}_{c',s'}[r + k, f']$, where $c' = 1 - \frac{1-c}{2^k}$, $s' = 1 - \frac{1-s}{2^k}$, and $f' = \log_2(2^{f-k} + 2^k - 1)$.*

The above can be further generalized, yet the current paper only utilizes the special case in which $c = 1$ (specifically, in the proof of part 3 of Proposition 10.9, we use $f = 2$ and $k = 1$ obtaining $f' = \log_2 3$, $c' = 1$, and $s' = \frac{1+s}{2}$).

Proof. The proof is similar to the proof of Proposition 11.8. Again, we consider a verifier V, as guaranteed by the hypothesis, and a random-pad $\overline{c}$, and let A_i be the set of (at most 2^{f-k}) accepting configurations which are consistent with the ith possibility of k oracle-answers to the first k queries. Denote the ith possibility by α_i (i.e., all configurations in A_i start with α_i). We construct a new verifier, V', which uniformly selects a random-pad $\overline{c}$ for V and $i \in [2^k]$ (specifying a part A_i as above). The verifier V' makes the first k queries of V and, if the answers differ from α_i, then V' halts and accepts.[18] Otherwise, V' continues the emulation of V and accepts iff V accepts.

Clearly, V' uses $r + k$ coin tosses. The accepting configurations of V' on random-pad $(\overline{c}, i)$ are those in A_i, as well as the "truncated V configurations" α_j, for $j \neq i$. Thus, there are at most $2^{f-k} + 2^k - 1$ accepting configurations. Suppose $V^\pi(x)$ accepts with probability p; then V' accepts input x with oracle access to π with probability $(1 - 2^{-k}) + 2^{-k} \cdot p = 1 - \frac{1-p}{2^k}$. The proposition follows. □

Finally, we present a simplified version of the above transformation. Here the acceptance probabilities are increased without affecting the free-bit complexity (either way).

PROPOSITION 11.10 (increasing acceptance probabilities while preserving free-bit complexity). *Let $c, s, \delta : \mathcal{Z}^+ \to [0,1]$ be admissible functions and $r, f : \mathcal{Z}^+ \to \mathcal{Z}^+$. Then*

$$\text{FPCP}_{c,s}[r, f] \subseteq \text{FPCP}_{c',s'}[r + \log(1/\delta), f],$$

where $c' = 1 - \delta \cdot (1 - c)$ and $s' = 1 - \delta \cdot (1 - s)$.

[18] In contrast, the verifier constructed in the proof of Proposition 11.8 rejects in case of such a mismatch.

Proof. Let V be a verifier for $L \in \mathrm{FPCP}_{c,s}[r,f]$. We construct a new verifier, V', which with probability δ invokes V and otherwise accepts regardless of the input. The proposition follows. □

Appendix. The coding theory bound. We provide here the coding theory bound used in the proof of Lemma 3.11. It is a slight extension of bounds in [73, Ch. 17] which consider only vectors of weight exactly w rather than at most w. For sake of completeness, we include a proof of this bound. In discussing binary vectors, the weight is the number of ones in the vector and the distance between two vectors is the number of places in which they disagree.

LEMMA A.1. *Let $B = B(n,d,w)$ be the maximum number of binary vectors of length n, each with weight at most w, and any two being distance at least d apart. Then $B \le (1-2\beta)/(4\alpha^2 - 2\beta)$, provided $\alpha^2 > \beta/2$, where $\alpha = (1/2) - (w/n)$ and $\beta = (1/2) - (d/n)$.*

Proof. Consider an arbitrary sequence $v_1, \ldots, v_M$ of n-vectors which are at mutual distance at least $n/2$. Let us denote by $v_{i,j}$ the jth entry in the ith vector, by w_i the weight of the ith vector, and by $\overline{w}$ the average value of the w_i's. Define

$$S \stackrel{\mathrm{def}}{=} \sum_{i=1}^{M}\sum_{j=1}^{M}\sum_{k=1}^{n} v_{i,k}v_{j,k}.$$

Then, on one hand,

$$\begin{aligned} S &= \sum_{i=1}^{M}\sum_{k=1}^{n} v_{i,k}^2 + \sum_{1\le i\ne j\le M}\sum_{k=1}^{n} v_{i,k}v_{j,k} \\ &\le \sum_i w_i + \sum_{1\le i\ne j\le M} \frac{w_i + w_j - d}{2} \\ &= M\overline{w} + M(M-1)\cdot(\overline{w} - (d/2)), \end{aligned}$$

where the inequality follows from observing that, for $i \ne j$,

$$\begin{aligned} w_i + w_j &= 2|\{k : v_{i,k} = v_{j,k} = 1\}| + |\{k : v_{i,k} \ne v_{j,k}\}| \\ &\ge 2\sum_{k=1}^{n} v_{i,k}v_{j,k} + d. \end{aligned}$$

On the other hand, $S = \sum_{k=1}^{n} |\{i : v_{i,k} = 1\}|^2$. This allows to lower bound S by the minimum of $\sum_k x_k^2$ subject to $\sum_k x_k = M\overline{w}$. The minimum is obtained when all x_k's are equal and yields

$$S \ge n \cdot \left(\frac{M\overline{w}}{n}\right)^2.$$

Confronting the two bounds, we get

$$\frac{M\cdot\overline{w}^2}{n} \le M\cdot\overline{w} - (M-1)\cdot\left(\frac{d}{2}\right)$$

which yields $(\frac{\overline{w}^2}{n} - \overline{w} + \frac{d}{2})M \le \frac{d}{2}$. Letting $\overline{\alpha} = (1/2) - (\overline{w}/n)$ and using

$$\overline{\alpha}^2 \ge \alpha^2 > \beta/2\,,$$

we get

$$M \leq \frac{1-2\beta}{4\overline{\alpha}^2 - 2\beta},$$

and the lemma follows by observing that the bound maximizes when $\alpha = \overline{\alpha}$. □

Acknowledgments. We thank Uri Feige, Johan Håstad, Viggo Kann, Marcos Kiwi, and Luca Trevisan for carefully reading previous versions of our work, pointing out several flaws, and suggesting improvements. We also thank two anonymous referees for their careful reading and many useful comments.

REFERENCES

[1] L. Adleman, *Two theorems on random polynomial time*, in Proceedings of the 19th Symposium on Foundations in Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1978, pp. 75–83.

[2] M. Ajtai, J. Komlos, and E. Szemeredi, *Deterministic simulation in logspace*, in Proceedings of the 19th Annual Symposium on the Theory of Computing, ACM, New York, 1987, pp. 132–140.

[3] N. Alon, U. Feige, A. Wigderson, D. Zuckerman, *Derandomized graph products*, Computat. Complexity, 5 (1995), pp. 60–75.

[4] N. Alon, J. Spencer, and P. Erdos, *The Probabilistic Method*, John Wiley and Sons, New York, 1992.

[5] E. Amaldi and V. Kann, *The complexity and approximability of finding maximum feasible subsystems of linear relations*, Theoret. Comput. Sci., 147 (1995), pp. 181–210.

[6] S. Arora, *Reductions, codes, PCPs, and inapproximability*, in Proceedings of the 36th Symposium on Foundations in Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 404–413.

[7] S. Arora, L. Babai, J. Stern, and Z. Sweedyk, *The hardness of approximate optima in lattices, codes and systems of linear equations*, J. Comput. System Sci., 54 (1997), pp. 317–331.

[8] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and intractability of approximation problems*, in Proceedings of the 33rd Symposium on Foundations in Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 14–23.

[9] S. Arora and S. Safra, *Probabilistic checking of proofs: A new characterization of NP*, in Proceedings of the 33rd Symposium on Foundations in Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 2–13.

[10] L. Babai, *Trading group theory for randomness*, in Proceedings of the 17th Annual Symposium on the Theory of Computing, ACM, New York, 1985, pp. 421–429.

[11] L. Babai, L. Fortnow, and C. Lund, *Non-deterministic exponential time has two-prover interactive protocols*, Computat. Complexity, 1 (1991), pp. 3–40. (See also addendum in 2 (1992), p. 374.)

[12] L. Babai, L. Fortnow, L. Levin, and M. Szegedy, *Checking computations in polylogarithmic time*, in Proceedings of the 23rd Annual Symposium on the Theory of Computing, ACM, New York, 1991, pp. 21–31.

[13] R. Bar-Yehuda and S. Even, *A linear time approximation algorithm for the weighted vertex cover problem*, J. Algorithms, 2 (1981), pp. 198–201.

[14] R. Bar-Yehuda and S. Even, *A local ratio theorem for approximating the weighted vertex cover problem*, in Analysis and Design of Algorithms for Combinatorial Problems, North-Holland Math. Stud. 109, North-Holland, Amsterdam, 1985, pp. 27–45.

[15] R. Bar-Yehuda and S. Moran, *On approximation problems related to the independent set and vertex cover problems*, Discrete Appl. Math., 9 (1984), pp. 1–10.

[16] M. Bellare, *Interactive proofs and approximation: Reductions from two provers in one round*, in Proceedings of the Second Israel Symposium on Theory and Computing Science, IEEE Computer Society Press, Los Alamitos, CA, 1993, pp. 266–274.

[17] M. Bellare, D. Coppersmith, J. Håstad, M. Kiwi, and M. Sudan, *Linearity testing in characteristic two*, IEEE Trans. Inform. Theory, 42 (1996), pp. 1781–1795.

[18] M. Bellare, O. Goldreich, and S. Goldwasser, *Randomness in interactive proofs*, Computat. Complexity, 3 (1993), pp. 319–354.

[19] M. Bellare, O. Goldreich, and M. Sudan, *Free bits, PCPs, and non-approximability—towards tight results*, extended abstract of this paper in Proceedings of the 36th Symposium on Foundations in Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 422–431.

[20] M. Bellare, O. Goldreich, and M. Sudan, *Free bits, PCPs, and non-approximability—towards tight results*, preliminary versions of this paper, Report TR95-024, *Elec. Colloq. Comput. Complexity*, May 1995 (revised Sept. 1995, Jan. 1996, Dec. 1996, and Aug. 1997), `http://www.eccc.uni-trier.de/eccc/`

[21] M. Bellare, S. Goldwasser, C. Lund, and A. Russell, *Efficient probabilistically checkable proofs and applications to approximation*, in Proceedings of the 25th Annual Symposium on the Theory of Computing, ACM, New York, 1993, pp. 294–304. (See also Errata sheet in Proceedings of the 26th Annual Symposium on the Theory of Computing, ACM, New York, 1994, pp. 820–820).

[22] M. Bellare and P. Rogaway, *The complexity of approximating a nonlinear program*, J. Math. Programming Ser. A, 69 (1995), pp. 429–441. Also in *Complexity of Numerical Optimization*, P. M. Pardalos, ed., World Scientific, River Edge, NJ, 1993, pp. 16–32.

[23] M. Bellare and M. Sudan, *Improved non-approximability results*, in Proceedings of the 26th Annual Symposium on the Theory of Computing, ACM, New York, 1994, pp. 184–193.

[24] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson, *Multi-prover interactive proofs: How to remove intractability assumptions*, in Proceedings of the 20th Annual Symposium on the Theory of Computing, ACM, New York, 1988, pp. 113–131.

[25] P. Berman and G. Schnitger, *On the complexity of approximating the independent set problem*, Inform. and Comput., 96 (1992), pp. 77–94.

[26] A. Blum, *Algorithms for Approximate Graph Coloring*, Ph.D. thesis, Department of Computer Science, MIT, Cambridge, MA, 1991.

[27] M. Blum, M. Luby, and R. Rubinfeld, *Self-testing/correcting with applications to numerical problems*, J. Comput. System Sci., 47 (1993), pp. 549–595.

[28] R. Boppana and M. Halldórsson, *Approximating maximum independent sets by excluding subgraphs*, BIT, 32 (1992), pp. 180–196.

[29] J. Bruck and M. Naor, *The hardness of decoding with preprocessing*, IEEE Trans. Inform. Theory, 36 (1990), pp. 381–385.

[30] A. Cohen and A. Wigderson, *Dispersers, deterministic amplification, and weak random sources*, in Proceedings of the 30th Symposium on Foundations in Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1989, pp. 14–19.

[31] A. Condon, *The complexity of the max word problem and the power of one-way interactive proof systems*, Comput. Complexity, 3 (1993), pp. 292–305.

[32] S. Cook, *The complexity of theorem-proving procedures*, in Proceedings of the 3rd Annual Symposium on the Theory of Computing, ACM, New York, 1971, pp. 151–158.

[33] P. Crescenzi and V. Kann, *A Compendium of NP Optimization Problems*, Technical Report, Dipartimento di Scienze dell'Informazione, Università di Roma "La Sapienza," SI/RR-95/02, 1995. The list is updated continuously. The latest version is available via `http://www.nada.kth.se/~viggo/problemlist/compendium.html`

[34] P. Crescenzi, R. Silvestri, and L. Trevisan, *To weight or not to weight: Where is the question*?, in Proceedings of the Fourth Israel Symposium on Theory of Computing and Systems, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 68–77.

[35] S. Even, A. Itai, and A. Shamir, *On the complexity of timetable and multicommodity flow problems*, SIAM J. Comput., 5 (1976), pp. 691–703.

[36] S. Even, A. Selman, and Y. Yacobi, *The complexity of promise problems with applications to public-key cryptography*, Inform. Control, 2 (1984), pp. 159–173.

[37] U. Feige, *Randomized graph products, chromatic numbers, and the Lovásztheta function*, in Proceedings of the 27th Annual Symposium on the Theory of Computing, ACM, New York, 1995, pp. 635–640.

[38] U. Feige, *Set cover. A threshold of ln n for approximating set cover*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, ACM, New York, 1996, pp. 314–318.

[39] U. Feige and M. Goemans, *Approximating the value of two prover proof systems, with application to Max-2SAT and Max-DICUT*, in Proceedings of the Third Israel Symposium on Theory and Computing Science, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 182–189.

[40] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, *Interactive proofs and the hardness of approximating cliques*, J. ACM, 43 (1996), pp. 268–292.

[41] U. Feige and J. Kilian, *Two prover protocols – low error at affordable rates*, in Proceedings of the 26th Annual Symposium on the Theory of Computing, ACM, New York, 1994, pp. 172–183.

[42] U. Feige and J. Kilian, *Zero-knowledge and the chromatic number*, in Proceedings of the 11th Annual Conference on Computational Complexity, IEEE Computer Society Press, Los Alamitos, CA, 1996.

[43] U. Feige and L. Lovász, *Two-prover one round proof systems: Their power and their problems*, in Proceedings of the 24th Annual Symposium on the Theory of Computing, ACM, New York, 1992, pp. 733–744.

[44] L. Fortnow, J. Rompel, and M. Sipser, *On the power of multiprover interactive protocols*, Theoret. Comput. Sci., 134 (1994), pp. 545–557.

[45] M. Fürer, *Improved hardness results for approximating the chromatic number*, in Proceedings of the 36th Symposium on Foundations in Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 414–421.

[46] M. Fürer, O. Goldreich, Y. Mansour, M. Sipser, and S. Zachos, *On completeness and soundness in interactive proof systems*, in Advances in Computing Research: A Research Annual, Randomness and Computation, S. Micali, ed., Vol. 5, JAI Press, Greenwich, CT, pp. 429–442.

[47] M. Garey and D. Johnson, *The complexity of near optimal graph coloring*, J. ACM, 23 (1976), pp. 43–49.

[48] M. Garey and D. Johnson, *Computers and Intractability: A guide to the Theory of NP-completeness*, W. H. Freeman and Company, San Francisco, CA, 1979.

[49] M. Garey, D. Johnson, and L. Stockmeyer, *Some simplified* NP*-complete graph problems*, Theoret. Comput. Sci., 1 (1976), pp. 237–267.

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

[51] M. Goemans and D. Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, J. ACM, 42 (1995), pp. 1115–1145.

[52] O. Goldreich, S. Micali, and A. Wigderson, *Proofs that yield nothing but their validity, or all languages in NP have zero-knowledge proof systems*, J. ACM, 38 (1991), pp. 691–729.

[53] S. Goldwasser, S. Micali, and C. Rackoff, *The knowledge complexity of interactive proofs*, SIAM J. Comput., 18 (1989), pp. 186–208.

[54] S. Goldwasser and M. Sipser, *Private coins versus public coins in interactive proof systems*, in Proceedings of the 18th Annual Symposium on the Theory of Computing, ACM, New York, 1986, pp. 59–68.

[55] J. Håstad, *Testing of the long code and hardness for clique*, in Proceedings of the 28th Annual Symposium on the Theory of Computing, ACM, New York, 1996, pp. 11–19.

[56] J. Håstad, *Clique is hard to approximate within* $n^{1-\epsilon}$, in Proceedings of the 37th Symposium on Foundations in Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 627–636.

[57] J. Håstad, *Getting optimal in-approximability results*, in Proceedings of the 29th Annual Symposium on the Theory of Computing, ACM, New York, 1997, pp. 1–10.

[58] D. Hochbaum, *Efficient algorithms for the stable set, vertex cover and set packing problems*, Discrete Appl. Math., 6 (1983), pp. 243–254.

[59] R. Impagliazzo and D. Zuckerman, *How to recycle random bits*, in Proceedings of the 30th Symposium on Foundations in Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1989, pp. 248–253.

[60] N. Jones, Y. Lien, and W. Laaser, *New problems complete for non-deterministic log space*, Math. Systems Theory, 10 (1976), pp. 1–17.

[61] N. Kahale, *Eigenvalues and expansion of regular graphs*, J. ACM, 42 (1995), pp. 1091–1106.

[62] V. Kann, S. Khanna, J. Lagergren, and A. Panconesi, *On the Hardness of Approximating MAX k-CUT and Its Dual*, Technical Report TRITA-NA-P9505, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, 1995.

[63] D. Karger, R. Motwani, and M. Sudan, *Approximate graph coloring by semidefinite programming*, in Proceedings of the 35th Symposium on Foundations in Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 2–13.

[64] H. Karloff and U. Zwick, *A* 7/8*-eps approximation algorithm for MAX* 3*SAT*?, in Proceedings of the 38th Symposium on Foundations in Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1997, pp. 406–415.

[65] R. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, Miller and Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.

[66] S. Khanna, N. Linial, and S. Safra, *On the hardness of approximating the chromatic number*, in Proceedings of the Second Israel Symposium on Theory and Computing Science, IEEE Computer Society Press, Los Alamitos, CA, 1993, pp. 250–260.

[67] D. Lapidot and A. Shamir, *Fully parallelized multi-prover protocols for NEXP-time*, in Proceedings of the 32nd Symposium on Foundations in Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1991, pp. 13–18.
[68] C. Lautemann, *BPP and the polynomial hierarchy*, Inform. Process Lett., 17 (1983), pp. 215–217.
[69] L. Levin, *Universal'nyĭe perebornyĭe zadachi (universal search problems)*, Problemy Peredachi Informatsii, 9 (1973), pp. 265–266, (in Russian).
[70] A. Lubotzky, R. Phillips, and P. Sarnak, *Explicit expanders and the Ramanujan conjectures*, in Proceedings of the 18th Annual Symposium on the Theory of Computing, ACM, New York, 1986, pp. 240–246.
[71] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, J. ACM, 41 (1994), pp. 960–981.
[72] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, *Algebraic methods for interactive proof systems*, J. ACM, 39 (1992), pp. 859–868.
[73] F. MacWilliams and N. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam, 1981.
[74] B. Monien and E. Speckenmeyer, *Ramsey numbers and an approximation algorithm for the vertex cover problem*, Acta Inform., 22 (1985), pp. 115–123.
[75] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, 1995.
[76] C. Papadimitriou and M. Yannakakis, *Optimization, approximation, and complexity classes*, J. Comput. System Sci., 43 (1991), pp. 425–440.
[77] E. Petrank, The Hardness of Approximations: Gap Location, Technical Report TR–754, Department of Computer Science, Technion – Israel Institute of Technology, 1992.
[78] A. Polishchuk and D. Spielman, *Nearly-linear size holographic proofs*, in Proceedings of the 26th Annual Symposium on the Theory of Computing, ACM, New York, 1994, pp. 194–203.
[79] R. Raz, *A parallel repetition theorem*, SIAM J. Comput., 27 (1998), pp. 763–803.
[80] S. Sahni and T. Gonzales, *P-complete approximation problems*, J. ACM, 23 (1976), pp. 555–565.
[81] A. Shamir, *IP=PSPACE*, J. ACM, 39 (1992), pp. 869–877.
[82] G. Tardos, *Multi-prover encoding schemes and three prover proof systems*, J. Comput. System Sci., 53 (1996), pp. 251–260.
[83] A. Ta-Shma, *A Note on PCP vs. MIP*, Inform. Process. Lett., 58 (1996), pp. 135–140.
[84] L. Trevisan, G. Sorkin, M. Sudan, and D. Williamson, *Gadgets, approximation and linear programming*, in Proceedings of the 37th Symposium on Foundations in Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 617–626.
[85] M. Yannakakis, *On the approximation of maximum satisfiability*, J. Algorithms, 17 (1994), pp. 475–502.
[86] D. Zuckerman, *On unapproximable versions of NP-complete problems*, SIAM J. Comput., 25 (1996), pp. 1293–1304.

SIAM J. COMPUT.
Vol. 27, No. 4, pp. 917–941, August 1998
© 1998 Society for Industrial and Applied Mathematics

FAST GOSSIPING BY SHORT MESSAGES*

JEAN-CLAUDE BERMOND†, LUISA GARGANO‡, ADELE A. RESCIGNO‡, AND UGO VACCARO‡

Abstract. Gossiping is the process of information diffusion in which each node of a network holds a packet that must be communicated to all other nodes in the network. We consider the problem of gossiping in communication networks under the restriction that communicating nodes can exchange up to a fixed number p of packets at each round. In the first part of the paper we study the extremal case $p = 1$ and we exactly determine the optimal number of communication rounds to perform gossiping for several classes of graphs, including Hamiltonian graphs and complete k-ary trees. For arbitrary graphs we give asymptotically matching upper and lower bounds. We also study the case of arbitrary p and we exactly determine the optimal number of communication rounds to perform gossiping under this hypothesis for complete graphs, hypercubes, rings, and paths. Finally, we investigate the problem of determining sparse networks in which gossiping can be performed in the minimum possible number of rounds.

Key words. gossiping, graphs, networks

AMS subject classifications. 94C15, 68M10

PII. S0097539795283619

1. Introduction.

Gossiping (also called total exchange or all-to-all communication) in distribution systems is the process of distributing information known to each processor to every other processor of the system. This process of information dissemination is carried out by means of a sequence of message transmissions between adjacent nodes in the network.

The gossiping problem was originally introduced by the community of discrete mathematicians, to which it owes most of its terminology, as a combinatorial problem in graphs. Nonetheless, it was soon realized that, once cast in more realistic models of communication, gossiping is a fundamental primitive in distributed memory multiprocessor systems. There are a number of situations in multiprocessor computation, such as global processor synchronization, where gossiping occurs. Moreover, the gossiping problem is implicit in a large class of parallel computation problems, such as linear system solving, the discrete Fourier transform, and sorting, where both input and output data are required to be distributed across the network [6, 9, 18]. Due to the interesting theoretical questions it poses and its numerous practical applications, gossiping has been widely studied under various communication models. Hedetniemi, Hedetniemi, and Liestman [15] provide a survey of the area. Two more recent survey papers collecting the latest results are [10, 17]. Readers could also profit from seeing the book [24].

* Received by the editors March 24, 1995; accepted for publication (in revised form) April 9, 1996; published electronically May 18, 1998. An extended abstract of this work was presented at *ICALP* '95, Szeged, Hungary.

http://www.siam.org/journals/sicomp/27-4/28361.html

† I3S, CNRS, Université de Nice, Sophia Antipolis, Bat ESSI, 650 Route de Colles, BP 145, 06903 Sophia Antipolis Cedex, France (bermond@alto.unice.fr). This research was partially supported by the French GDR/PRC Project PRS and by the French–Italian Bilateral Project Galileo. This research was partially performed while the author was visiting the Dipartimento di Informatica ed Applicazioni of the Università di Salerno.

‡ Dipartimento di Informatica ed Applicazioni, Università di Salerno, 84081 Baronissi (SA), Italy (lg@dia.unisa.it, rescigno@udsab.dia.unisa.it, uv@dia.unisa.it). This research was partially supported by the Italian Ministry of University and Scientific Research, Project: Algoritmi, Modelli di Calcolo e Strutture Informative and by the French–Italian Bilateral Project Galileo.

The great majority of the previous work on gossiping has considered the case in which the packets known to a processor at any given time during the execution of the gossiping protocol can be freely concatenated and the resulting (longer) message can be transmitted in a constant amount of time; that is, it has been assumed that the time required to transmit a message is independent of its length. While this assumption is reasonable for short messages, it is clearly unrealistic when the size of the messages becomes large. Notice that most of the gossiping protocols proposed in the literature require the transmission, in the last rounds of the execution of the protocol, of messages of size $\Theta(n)$, where n is the number of nodes in the network. Therefore, it would be interesting to have gossiping protocols that require only the transmission of bounded length messages between processors. In this paper we consider the problem of gossiping in communication networks under the restriction that communicating nodes can exchange up to a fixed number p of packets in each round.

1.1. The model. Consider a communication network modeled by a graph $G = (V, E)$ where the node set V represents the set of processors of the network and E represents the set of communication lines between processors.

Initially each node holds a packet that must be transmitted to any other node in the network by a sequence of *calls* between adjacent processors. During each call, communicating nodes can exchange up to p packets, where p is an a priori fixed integer. We assume that each processor can participate in at most one call at a time. Therefore, we can see the gossiping process as a sequence of *rounds*: during each round a disjoint set of edges (matching) is selected and the nodes that are end vertices of these edges make a call. This communication model is usually referred to as *telephone model* [15] or *full-duplex* 1-*port* (F_1) [20]. We denote by $g_{F_1}(p, G)$ the minimum possible number of rounds to complete the gossiping process in the network G subject to the above conditions. Another popular communication model is the *mail model* [15] or *half-duplex* 1-*port* (H_1) [20], in which in each round any node can either send a message to one of its neighbors or receive a message from it, but not simultaneously. The problem of estimating $g_{H_1}(p, G)$ has been considered in [4]. Analogous problems in bus networks have been considered in [11, 16]. Optimal bounds on $g_{H_1}(1, G)$ when the edges of G are subject to random failures are given in [7]. Packet routing in interconnection networks in the F_1 model has been considered in [1].

1.2. Results. We first study the extremal case in which gossiping is to be performed under the restriction that communicating nodes can exchange *exactly* one packet in each round. We provide several lower bounds on the gossiping time $g_{F_1}(1, G)$ and we provide matching upper bounds for Hamiltonian graphs, complete trees, and complete bipartite graphs. For general graphs we provide asymptotically tight upper and lower bounds.

Subsequently, we study the case of arbitrary p and we compute $g_{F_1}(p, G)$ exactly for complete graphs, hypercubes, rings, and paths. Our result for hypercubes allows us to improve the corresponding result in the H_1 model given in [4].

Finally, we investigate the problem of finding the sparsest networks in which gossiping can be performed in the minimum possible number of rounds.

All logarithms in this paper are to the base 2.

2. Gossiping by exchanging one packet at a time. In this section we study $g_{F_1}(1, G)$, that is, the minimum possible number of rounds to complete gossip in a graph G under the condition that at each round communicating nodes can exchange

exactly one packet.

In order to avoid overburdening the notation, we will simply write $g(G)$ to denote $g_{F_1}(1, G)$.

2.1. Lower bounds on $g(G)$. In this section we give some lower bounds on the time needed to complete the gossiping process.

LEMMA 2.1. *For any graph $G = (V, E)$, with $|V| = n$, let $\mu(G)$ be the size of a maximum matching in G; then*

$$g(G) \geq \left\lceil \frac{n(n-1)}{2\mu(G)} \right\rceil. \tag{1}$$

Proof. For any node $v \in V$ the packet initially resident in v must reach each of the remaining $n-1$ nodes. Therefore, during the gossiping process, at least $n(n-1)$ packet transmissions must be executed over the edges of G. Since in each communication round at most $\mu(G)$ calls are performed and each call allows the transmission of two packets (one in each direction), the bound follows. □

LEMMA 2.2. *Let $X \subset V$ be a vertex cutset of the graph $G = (V, E)$ whose removal disconnects G into the connected components $V_1, \ldots, V_d$; then*

$$g(G) \geq \left\lceil \sum_{i=1}^{d} \frac{\max\{|V_i|, n - |V_i|\}}{|M_X|} \right\rceil, \tag{2}$$

where $|M_X|$ is the size of a maximum matching M_X in G such that any edge in it has one endpoint in X and the other in $V - X$.

Proof. Consider a component V_i, for some $1 \leq i \leq d$. Nodes in V_i can receive the packets of nodes in $V - V_i$ only by means of calls between a node in X and one in V_i; moreover, at least $n - |V_i|$ calls are needed between nodes in X and nodes in V_i to bring all packets in $V - V_i$ to nodes in V_i. Analogously, packets of nodes in V_i can reach nodes in $V - V_i$ only by means of calls between a node in X and one in V_i, and at least $|V_i|$ such calls are needed. Therefore, for each $i = 1, \ldots, d$, at least $\max\{|V_i|, n - |V_i|\}$ calls must take place between nodes in X and nodes in V_i. We then get that at least $\sum_{i=1}^{d} \max\{|V_i|, n - |V_i|\}$ calls are needed between nodes in X and nodes in $V - X = \cup_{i=1}^{d} V_i$. Since at most $|M_X|$ such calls can take place during each round, we get the desired lower bound of

$$\left\lceil \frac{\sum_{i=1}^{d} \max\{|V_i|, n - |V_i|\}}{|M_X|} \right\rceil$$

on the time necessary to gossip in G. □

Remark 2.1. The bound in Lemma 2.2 can sometimes be improved by observing that after the last call has been done between a node in some V_i and a node in X, the last exchanged message has still to reach all the other nodes of V_i (or of $V - V_i$). Therefore, we can add to the lower bound (2) the minimum of the eccentricities of the subgraphs induced by the V_i's and the $V - V_i$'s.

COROLLARY 2.1. *Let $\alpha(G)$ be the independence number of G; then*

$$g(G) \geq \left\lceil \frac{\alpha(G)(n-1)}{n - \alpha(G)} \right\rceil. \tag{3}$$

Proof. Let Y denote an independent set of G. Applying Lemma 2.2 with cutset $X = V - Y$ and connected components $V_1, \ldots, V_{|Y|}$, each consisting of just one element of Y, we get

$$g(G) \geq \left\lceil \sum_{i=1}^{|Y|} \frac{n - |V_i|}{|M_X|} \right\rceil \geq \left\lceil \sum_{i=1}^{|Y|} \frac{n - |V_i|}{|X|} \right\rceil = \left\lceil \sum_{i=1}^{|Y|} \frac{n-1}{n - |Y|} \right\rceil = \left\lceil \frac{|Y|(n-1)}{n - |Y|} \right\rceil.$$

Choosing an independent set of maximum size $|Y| = \alpha(G)$ we get (3). □

Let T be a tree and v one of its nodes; we indicate the connected components into which the node set of T is split by the removal of v by $V_1(v), \ldots, V_{\deg(v)}(v)$, ordered so that $|V_1(v)| \geq \cdots \geq |V_{\deg(v)}(v)|$.

COROLLARY 2.2. *Let T be a tree on n nodes of maximum degree $\Delta = \max_{v \in V} \deg(v)$; then*

$$g(T) \geq \max_{v\,:\,\deg(v)=\Delta} L(v),$$

where

$$L(v) = \begin{cases} (\deg(v) - 1)n + 1 & \text{if } |V_1(v)| \leq n/2, \\ (\deg(v) - 2)n + 1 + 2|V_1(v)| & \text{if } |V_1(v)| > n/2. \end{cases}$$

Proof. Given a node v, Lemma 2.2 with $X = \{v\}$ gives

$$\begin{aligned} g(T) &\geq \sum_{i=1}^{\deg(v)} \max\{|V_i(v)|, n - |V_i(v)|\} \\ &= \sum_{i=2}^{\deg(v)} n - |V_i(v)| + \begin{cases} |V_1(v)| & \text{if } |V_1(v)| > n/2, \\ n - |V_1(v)| & \text{if } |V_1(v)| \leq n/2, \end{cases} \\ &= L(v). \end{aligned}$$

Direct computation shows that if $\deg(v) > \deg(w)$ then $L(v) > L(w)$, thus proving that the maximum is always attained at a node of maximum degree. □

2.2. Upper bounds. In this section we will determine $g(G)$ exactly for several classes of graphs, including Hamiltonian graphs and complete k-ary trees. We will also provide good upper bounds for general graphs.

2.2.1. Hamiltonian graphs. We first note that in any graph $G = (V, E)$ the size of a maximum matching $\mu(G)$ is at most $\lfloor |V|/2 \rfloor$. Therefore, from Lemma 2.1 we get that the gossiping time $g(G)$ of *any* graph with n nodes is always lower-bounded by

$$g(G) \geq \begin{cases} n - 1 & \text{if } n \text{ is even}, \\ n & \text{if } n \text{ is odd}. \end{cases} \tag{4}$$

We will show that this lower bound is attained by Hamiltonian graphs.

Let $C_n = (V, E)$ denote the ring of length n; we assume that the vertex set is $V = \{0, \ldots, n-1\}$ and the edge set is $E = \{(v, w) \ : \ 1 = |v - w| (\text{mod } n)\}$.[1]

[1] Here and in the rest of the paper with $x = a \pmod b$ we denote the unique integer $0 \leq x < b$ such that $x = qb + a$.

Gossiping-even(C_n)
Round $t = 1$**:** Each node v sends its own packet to the node w such that $(v, w) \in M_1$.
Round $t = 2$**:** Each node v sends its own packet to the node w such that $(v, w) \in M_2$.
Round t**,** $3 \leq t \leq n-1$**:** For each node v let w be the node such that $(v, w) \in M_t$; node v sends a new packet to w; namely, v sends the packet it first received from among those v has neither received from w nor sent to w in any previous round.

FIG. 1. *Gossiping algorithm in C_n, n even.*

Gossiping-odd(C_n)
Round t**,** $1 \leq t \leq n$**:** For each node $v \neq t-1$ let w be the neighbor of v in M_t; node v sends to w the packet that v first received from among those that it neither got from w nor sent to w in a previous round (v's own packet is considered to be received before any other packet).

FIG. 2. *Gossiping algorithm in C_n, n odd.*

LEMMA 2.3. $$g(C_n) \leq \begin{cases} n-1 & \text{if } n \text{ is even,} \\ n & \text{if } n \text{ is odd.} \end{cases}$$

Proof. We distinguish two cases according to the parity of the number n of nodes.

Case n even. We shall give a gossiping protocol on the ring C_n that requires $n-1$ rounds. First, for each integer t, define the perfect matching in C_n given by

$$M_t = \begin{cases} \{(v, w) \ : \ v \text{ is even and } w = v+1\} & \text{if } t \text{ is even,} \\ \{(v, w) \ : \ v \text{ is odd and } w = v+1 (\text{mod } n)\} & \text{if } t \text{ is odd;} \end{cases} \tag{5}$$

notice that M_t and M_{t+1} are disjoint for each t. The gossiping algorithm is shown in Figure 1.

It is immediate to see that each node receives a new packet at each round (this can be formally proved by induction on t). Therefore, at the end of round $n-1$ of algorithm **Gossiping-even**(C_n) each node has received all the packets of the other $n-1$ nodes.

Case n odd. Define the following maximum matchings M_t in C_n for each $t = 1, \ldots, n$:

$$M_t = \{(v, w) \ : \ v - t + 1 \, (\text{mod } n) \text{ is odd, } w = v + 1 (\text{mod } n), \text{ and } v \neq t-1 \neq w\}. \tag{6}$$

We give in Figure 2 a gossiping protocol on C_n that requires n rounds. It is easy to see that at each round $t = 1, \ldots, n$ each node different from $t-1$ receives a new packet. Therefore, at the end of round n of algorithm **Gossiping-odd**(C_n), each node has received all the packets of the other $n-1$ nodes. □

Example 2.1. For $n = 6$ we have $M_1 = M_3 = M_5 = \{(1, 2), (3, 4), (50)\}$ and $M_2 = M_4 = \{(0, 1), (2, 3), (4, 5)\}$. Each column of Table 1 shows the set of nodes whose packets are known by v at the end of round t, for each $0 \leq v \leq 5$ and $1 \leq t \leq 5$.

TABLE 1

$t\backslash v$	0	1	2
1	$\{5,0\}$	$\{1,2\}$	$\{1,2\}$
2	$\{5,0,1\}$	$\{0,1,2\}$	$\{1,2,3\}$
3	$\{4,5,0,1\}$	$\{0,1,2,3\}$	$\{0,1,2,3\}$
4	$\{4,5,0,1,2\}$	$\{5,0,1,2,3\}$	$\{0,1,2,3,4\}$
5	$\{3,4,5,0,1,2\}$	$\{5,0,1,2,3,4\}$	$\{5,0,1,2,3,4\}$

$t\backslash v$	3	4	5
1	$\{3,4\}$	$\{3,4\}$	$\{5,0\}$
2	$\{2,3,4\}$	$\{3,4,5\}$	$\{4,5,0\}$
3	$\{2,3,4,5\}$	$\{2,3,4,5\}$	$\{4,5,0,1\}$
4	$\{1,2,3,4,5\}$	$\{2,3,4,5,0\}$	$\{3,4,5,0,1\}$
5	$\{1,2,3,4,5,0\}$	$\{1,2,3,4,5,0\}$	$\{3,4,5,0,1,2\}$

TABLE 2

$t\backslash v$	0	1	2	3	4
1	$\{0\}$	$\{1,2\}$	$\{1,2\}$	$\{3,4\}$	$\{3,4\}$
2	$\{4,0\}$	$\{1,2\}$	$\{1,2,3\}$	$\{2,3,4\}$	$\{3,4,0\}$
3	$\{4,0,1\}$	$\{0,1,2\}$	$\{1,2,3\}$	$\{2,3,4,0\}$	$\{2,3,4,0\}$
4	$\{3,4,0,1\}$	$\{0,1,2,3\}$	$\{0,1,2,3\}$	$\{2,3,4,0\}$	$\{2,3,4,0,1\}$
5	$\{3,4,0,1,2\}$	$\{0,1,2,3,4\}$	$\{0,1,2,3,4\}$	$\{1,2,3,4,0\}$	$\{2,3,4,0,1\}$

For $n = 5$ we have $M_1 = \{(1,2),(3,4)\}$, $M_2 = \{(2,3),(4,0)\}$, $M_3 = \{(3,4),(0,1)\}$ $M_4 = \{(4,0),(1,2)\}$, and $M_5 = \{(0,1),(2,3)\}$. Each column of Table 2 shows the set of nodes whose packets are known by v at the end of round t, for each $0 \le v \le 4$ and $1 \le t \le 5$.

THEOREM 2.1. *For any Hamiltonian graph G on n vertices,*

$$g(G) = \begin{cases} n-1 & \text{if } n \text{ is even,} \\ n & \text{if } n \text{ is odd.} \end{cases}$$

Proof. If G is Hamiltonian, from Lemma 2.3 we get that gossiping along the edges of the Hamiltonian cycle requires time matching the lower bound (4), and the theorem holds. □

2.3. Trees. In this section we investigate the gossiping time in trees. We first give an upper bound on the gossiping time in *any* tree and afterwards compute the exact gossiping time of rooted k-ary trees.

Consider a tree $T = (V, E)$. We recall that for each node v the set $V_1(v)$ denotes the largest of the connected components into which T splits after the removal of v. Let

$$\vartheta = \max |V_1(v)|,$$

where the maximum is taken over all the internal nodes v having exactly $\deg(v) - 1$ leaves as neighbors; notice that any internal node u with less than $d(u) - 1$ leaves as neighbors has $|V_1(u)| \le \vartheta - 1$.

Let *pre-leaf* denote a node v such that $|V_1(v)| = \vartheta$ and denote by π the maximum degree of a node in the subgraph consisting only of the edges (u, f) where f is either a leaf or a pre-leaf of T. Finally, let λ be the maximum number of leaves connected to a common node and $\Delta = \max_{v \in V} \deg(v)$.

Gossiping-tree(T)
Phase 1
[*Color each edge* (u,v) *of* T *with color* $c(u,v)=c(v,u)\in\{0,\dots,\Delta-1\}$.]
Round t**, for** $t=1,\dots,\Delta(\vartheta-1)$**:** For each node u, if there is an edge (u,v) such that $c(u,v)=t-1(\mathrm{mod}\ \Delta)$, then u sends a new packet to v; namely, u sends to v a packet from among those that u has neither sent to v nor received from v in a previous round, if such a packet exists; otherwise u sends nothing.
Phase 2
[*Give to each edge* (u,f)*, where* f *is a leaf or a pre-leaf, a color* $c'(u,f)\in\{0,\dots,\pi-1\}$.]
Round $\Delta(\vartheta-1)+t$**, for** $t=1,\dots,\pi$**:** For each leaf or pre-leaf f, if there is an edge (u,f) with $c'(u,f)=t-1$, then u sends to f a packet from among those that u has neither sent to f nor received from f in a previous round, if any.
Phase 3
[*Give to each edge* (u,f)*, where* f *is a leaf of* T*, a color* $c''(u,f)\in\{0,\dots,\lambda-1\}$.]
Round $\Delta(\vartheta-1)+\pi+t$**, for** $t=1,\dots,(n-\vartheta-1)\lambda$**:** For each leaf f, if the edge (u,f) on f has $c''(u,f)=t-1(\mathrm{mod}\ \lambda)$, then u sends to f any packet f does not know.

FIG. 3. *Gossiping algorithm in a tree* T.

THEOREM 2.2. *For any tree* $T=(V,E)$ *on* n *nodes,* $g(T)\le(\vartheta-1)\Delta+\pi+(n-\vartheta-1)\lambda$.

Proof. Consider the gossiping algorithm **Gossiping-tree**(T) given in Figure 3. For any $e\in E$, the edge coloring $c(e)$ and the partial edge colorings $c'(e)$ and $c''(e)$ used in **Gossiping-tree**(T) are each intended so that no two edges sharing a vertex are assigned the same color.

We now prove the correctness of algorithm **Gossiping-tree**(T). Let us say that the edge (u,v) of T is *saturated* from u to v at time t of **Gossiping-tree**(T) if no packet is sent from u to v at *any time* $t'\ge t$, that is, if by time $t-1$ node v has received the packet of each node w connected to v through u. We need the following property of **Gossiping-tree**(T).

PROPERTY 2.1. *In any round* t *of Phase* 1 *of the algorithm* **Gossiping-tree**(T)*, if the edge* (u,v) *has color* $c(u,v)=t-1(\mathrm{mod}\ \Delta)$ *and it is not saturated from* u *to* v *at time* t*, then* u *sends a new packet to* v *at round* t.

Proof. The proof is by induction on the time unit t. Let $t\le\Delta$: at time unit t, for each edge (u,v) of color $c(u,v)=t-1\in\{0,\dots,\Delta-1\}$, nodes u and v exchange a call for the first time and have at least their own packet to send each other.

Now let $t>\Delta$ and suppose that the hypothesis holds for each $t'<t$.

Consider an edge (u,v) such that $c(u,v)=t-1(\mathrm{mod}\ \Delta)$. Suppose by contradiction that at time t the edge (u,v) is not saturated from u to v, but u has no packets to send to v among those u has not received through v. That is, all packets known to u and not received through v have already been sent from u to v.

In particular, node u has already sent to v all the packets it has received from its other neighbors, call them $w_1,\dots,w_k$. Notice that the last call from u to v has taken place at time $t-\Delta$. For each w_i, let τ_i be the only integer such that both

$t-\Delta < \tau_i < t$ and $c(u,w_i) = \tau_i - 1 (\mathrm{mod}\ \Delta)$ hold. If the edge (u,w_i) is not saturated at time τ_i, we know by the inductive hypothesis that u has received a packet from w_i at time τ_i. We can have two cases: the first case is that all the edges (u,w_i) are saturated at time $\tau_i < t$. This immediately implies that (u,v) is saturated at time t, contradicting our assumption that (u,v) was not saturated from u to v at time t. The second case is that at least one edge (u,w_i) is not saturated at time τ_i; in such a situation we know by the inductive hypothesis that u has received a new packet from w_i at time τ_i that can now be forwarded to v, again getting a contradiction. □

We can now complete the proof of the theorem by showing that at the end of **Gossiping-tree**(T) each node knows all the other $n-1$ packets. Property 2.1 shows that a new packet is sent from u to v at each round t of **Phase 1** such that $c(u,v) = t-1 \ (\mathrm{mod}\ \Delta)$, until the edge (u,v) is saturated and no more packets need to be sent from u to v. Therefore, for any internal node u and for any Δ consecutive rounds, u receives a new packet from each neighbor v such that (v,u) is not saturated from v to u. We recall that ϑ is the maximum number of packets that any internal node needs to get from one neighbor and that this maximum is attained with equality only if u is a pre-leaf. Therefore, by round $\Delta(\vartheta-1)$, any node u which is not a pre-leaf gets all the necessary $n-1$ packets, while a pre-leaf gets $n-2$ packets during the $\Delta(\vartheta-1)$ rounds of Phase 1, and the remaining packet during some round of Phase 2.

Analogously, during Phase 1 any leaf f gets $\vartheta-1$ packets. It is obvious that f receives a new packet during Phase 2; moreover, during Phase 3 the leaf f receives a new packet for any λ consecutive rounds, thus getting the remaining $n-\vartheta-1$ packets that it needs to complete the gossip. □

Let δ denote the minimum degree of an internal node in T. It is easy to see that we can upper-bound ϑ by $n-\delta$. Therefore, from Theorem 2.2 we have the following upper bound on $g(T)$ that is expressed only in terms of degree properties of the nodes in T.

COROLLARY 2.3. *For any tree $T=(V,E)$ on n nodes, $g(T) \le (n-\delta)\Delta + (\delta-1)\lambda$.*

Given a connected graph $G=(V,E)$, denote by $\mathcal{T}$ the set of all spanning trees of G, and for any vertex $v \in V$, denote by $\deg_T(v)$ the degree of v in $T \in \mathcal{T}$. Define $d(G) = \min_{T\in\mathcal{T}} \max_{v\in V} \deg_T(v)$. The following corollary is immediate.

COROLLARY 2.4. *For any connected graph $G=(V,E)$ with n vertices,*

$$g(G) \le (n-1)d(G). \tag{7}$$

We point out that, although the problem of computing $d(G)$ is NP-hard, there exists an efficient algorithm to compute a spanning tree of maximum degree at most $d(G)+1$ (see [12]). From Corollaries 2.2 and 2.4 we have that for any tree with n nodes and maximum degree Δ it holds that $n\Delta - n + 1 \le g(T) \le n\Delta - \Delta$. Let us now consider the tree $S_{n,\Delta}$ of Figure 4. If $\Delta = n-1$, then $S_{n,n-1}$ is the star on n nodes, and from Corollary 2.2 and Theorem 2.2 we have $g(S_{n,n-1}) = (n-1)^2$. If $\Delta > 2$ is constant with respect to $n > 2\Delta$ then from Corollary 2.2 and Theorem 2.2 we get $\Delta(n-1) - (\Delta-1) \le g(S_{n,\Delta}) \le \Delta(n-1) - 2$. It is not difficult to obtain a specific gossiping algorithm attaining the lower bound, hence the bound (7) is asymptotically tight.

In [7] it was conjectured that for any G, $g_{H_1}(1,G) = \Omega(n\ d(G))$ holds; Ravi [22] has proved the following theorem.

THEOREM 2.3 (see [22]). *For any graph G, $g(G) = \Theta(n\ d(G))$.*

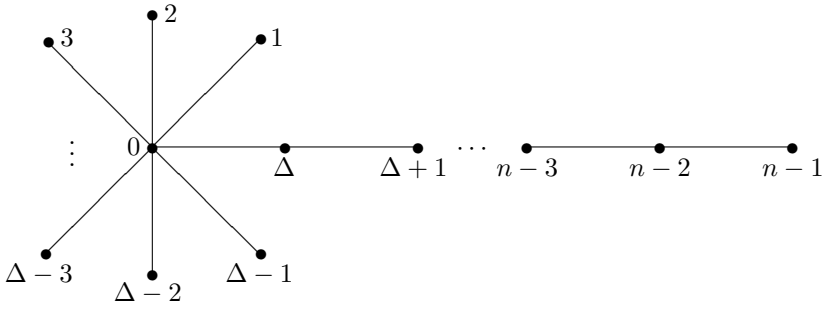


FIG. 4. *Tree* $S_{n,\Delta}$.

Proof. Given a vertex cutset X of $G = (V, E)$, from Lemma 2.2 we know that

$$g(G) \geq \left\lceil \sum_{i=1}^{c(X)} \frac{\max\{|V_i|, n - |V_i|\}}{|M_X|} \right\rceil,$$

where $V_1, \ldots, V_{c(X)}$ are the connected components induced in G by $V - X$ and M_X is a maximum matching between X and $V - X$ in G. Noticing that $|M_X| \leq |X|$ and $\max\{|V_i|, n - |V_i|\} \geq n/2$ for each i,

$$g(G) \geq \frac{n}{2} \frac{c(X)}{|X|}. \tag{8}$$

It was proved by Fürer and Raghavachari (see [12, Section 5]) that there exists a vertex cutset $Y \subset V$ such that

$$d(G) \leq \frac{c(Y)}{|Y|} + 1.$$

Using the above inequality and the lower bound (8), we have

$$g(G) \geq \frac{n}{2} \left(d(G) - 1 \right).$$

The above inequality, together with Corollary 2.4, implies that $g(G) = \Theta(n\ d(G))$. □

We remark that the same reasoning as in Theorem 2.3 allows us to prove that for any p

$$g_{F_1}(p, G) = \Omega(n\ d(G)/p).$$

We shall now compute the exact gossiping time of k-ary trees, that is, rooted trees in which each internal node has exactly k sons. Let $T_{k,n}$ denote *any* k-ary tree with n nodes.

Let us first notice that for $n = k + 1$ the tree $T_{k,n}$ is the star $S_{k+1,k}$. Consider then a tree $T_{k,n}$ with $n \geq 2k + 1$ nodes. Let u be a node of $T_{k,n}$ whose sons are all leaves. By Corollary 2.2 we get

$$g(T_{k,n}) \geq \max_v L(v) \geq L(u) = \begin{cases} kn + 1 & \text{if } n = 2k + 1, \\ (k + 1)(n - 1) - k & \text{if } n \geq 3k + 1. \end{cases} \tag{9}$$

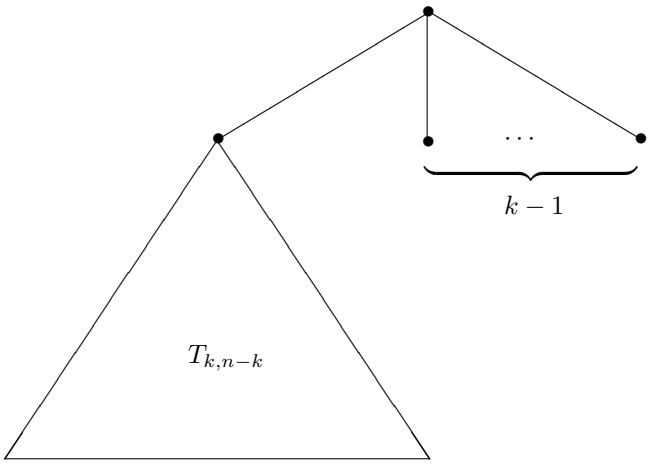


FIG. 5.

We show now that (9) holds with equality. Applying Theorem 2.2 to $T_{k,n}$ we get that

$$g(T_{k,n}) \leq (\vartheta - 1)\Delta + \pi + (n - \vartheta - 1)\lambda = (\vartheta - 1)(k+1) + \pi + (n - \vartheta - 1)k. \tag{10}$$

Unless exactly $k-1$ sons of the root are leaves (cf. the tree in Figure 5), $T_{k,n}$ has $\vartheta = n - k - 1$ and $\pi \leq \Delta = k + 1$; that, by (10) and (9), gives

$$g(T_{k,n}) = (n-k-2)(k+1) + k + 1 + k^2 = (k+1)(n-1) - k.$$

Consider the remaining case when $T_{k,n}$ is the tree of Figure 5. The only pre-leaf is the root, and $\vartheta = n - k$. If $n \geq 3k + 1$ we have $\pi = k$, and from (10) we get

$$g(T_{k,n}) \leq (n-k-1)(k+1) + k + (k-1)k = (n-1)(k+1) - k;$$

if $n = 2k + 1$ we have $\pi = \Delta = k + 1$ and $g(T_{k,2k+1}) \leq kn + 1$.

Therefore, we have proved the following result.

THEOREM 2.4. *For any k-ary tree on n nodes $T_{k,n}$,*

$$g(T_{k,n}) = \begin{cases} k^2 & \text{if } n = k+1, \\ 2k^2 + k + 1 & \text{if } n = 2k+1, \\ (k+1)(n-1) - k & \text{if } n \geq 3k+1. \end{cases}$$

The particular case $k = 1$ of the above result deserves to be explicitly stated.

COROLLARY 2.5. *Let P_n be the path on n nodes. We have*

$$g(P_n) = \begin{cases} 1 & \text{if } n = 2, \\ 4 & \text{if } n = 3, \\ 2n - 3 & \text{if } n \geq 4. \end{cases}$$

2.4. Complete bipartite graphs. Let $K_{r,s} = (V(K_{r,s}), E(K_{r,s}))$ be the complete bipartite graph on the node set $V(K_{r,s}) = \{a_0, \ldots, a_{r-1}\} \cup \{b_0, \ldots, b_{s-1}\}$, with $r \geq s$, $\{a_1, \ldots, a_{r-1}\} \cap \{b_0, \ldots, b_{s-1}\} = \emptyset$, and edge set $E(K_{r,s}) = \{a_0, \ldots, a_{r-1}\} \times \{b_0, \ldots, b_{s-1}\}$. In the next theorem we determine the gossiping time of $K_{r,s}$.

THEOREM 2.5. *For each r and s with $r \geq s \geq 1$, it holds that* $g(K_{r,s}) = \lceil (r+s-1)r/s \rceil$.

Proof. The lower bound $g(K_{r,s}) \geq \lceil (r+s-1)r/s \rceil$ is an immediate consequence of Corollary 2.1 since the complete bipartite graph has $\alpha(K_{r,s}) = r$.

In order to give a gossiping algorithm in $K_{r,s}$ requiring $\lceil (r+s-1)r/s \rceil$ communication rounds, we define the matchings

$$M_j = \{(b_i, a_{i+j \pmod r}) \ : \ 0 \leq i \leq s-1\}$$

for $j = 0, \ldots, r-1$. The algorithm is shown in Figure 6.

According to the protocol, at the end of **Phase 1** of **Gossiping-bipartite**$(K_{r,s})$ each node a_i (resp., b_i) knows the message of each b_i (resp., a_i). Consider now **Phase 2**. It is immediate to see that during the first $s-1$ rounds of **Phase 2** each of the b_i's receives the packet of each b_j for $j \neq i$, thus completing its knowledge. Moreover, after the $\lceil r(r+s-1)/s \rceil - r = \lceil r(r-1)/s \rceil$ rounds of **Phase 2** each node a_i has been involved in a call at least $r-1$ times and has then received the packet of each of the a_j, for $j \neq i$, thus completing its knowledge. □

Gossiping-bipartite$(K_{r,s})$
Phase 1
round t**, for** $t = 1, \ldots, r$**:** For each edge $(b_i, a_{i+t-1 \pmod r}) \in M_{t-1}$ nodes b_i and $a_{i+t-1 \pmod r}$ exchange their own packets.
Phase 2
round t**, for** $t = r+1, \ldots, \lceil r(r+s-1)/s \rceil$**:**
For each edge $(b_i, a_j) \in M_{(t-1-r)s \pmod r}$ node b_i sends to a_j any packet that a_j has not received in a previous round;
if $t \leq r+s-1$ then b_i receives from a_j the packet of $b_{i+t-r \pmod s}$.

FIG. 6. *Gossiping algorithm in $K_{r,s}$.*

2.5. Generalized Petersen graphs. In section 2.2.1 we have seen that Hamiltonian graphs have the minimum possible gossiping time among all graphs with n nodes. It is natural to wonder whether there are non-Hamiltonian graphs on n vertices with gossiping time equal to n if n is odd and $n-1$ if n is even. A quick check shows that this is not the case for rectangular grids $G_{t,s}$ with both t and s odd.[2] In fact, we know that $\alpha(G_{t,s}) = \lceil \frac{s \cdot t}{2} \rceil$, and from Corollary 2.1, we get $g(G_{t,s}) \geq s \cdot t + 1$. Moreover, it is also easy to check that the gossiping time of the Petersen graph on 10 vertices is at least 10. Therefore, one could be tempted to conjecture that the gossiping time $g(G)$ of a graph G is equal to the minimum possible only if G is Hamiltonian. This conjecture, although nice sounding, would be wrong, as the following classes of graphs, including the generalized Petersen graphs (GPGs), show.

Let $P_{k,\pi}$ be the graph consisting of two cycles of size k connected by a perfect matching in the following way: given a permutation π of $\{0, \ldots, k-1\}$ the graph $P_{k,\pi} = (V(P_{k,\pi}), E(P_{k,\pi}))$ has vertex set $V(P_{k,\pi}) = \{a_0, \ldots, a_{k-1}\} \cup \{b_0, \ldots, b_{k-1}\}$ and edge set

[2] It is well known that all rectangular grids $G_{t,s}$ are Hamiltonian except when t and s are both odd.

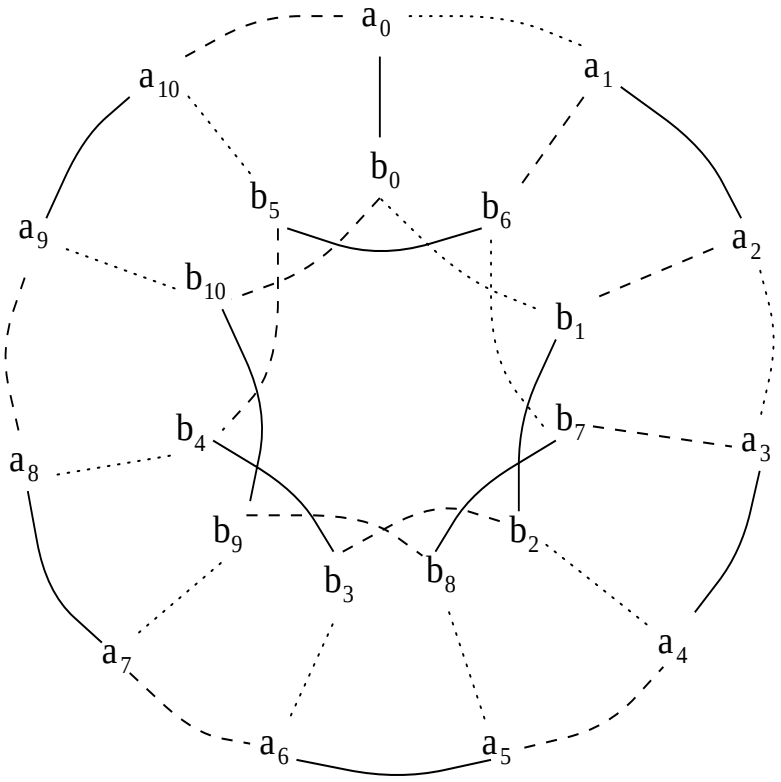


FIG. 7. *A 3-coloration of the GPG with* $n = 11$ *and* $s = 2$.

$$E(P_{k,\pi}) = \{(a_i, a_{i+1(\mathrm{mod}\ k)}) \ : \ 0 \le i < k\}$$
$$\cup \{(b_i, b_{i+1(\mathrm{mod}\ k)}) \ : \ 0 \le i < k\} \cup \{(a_i, b_{\pi(i)}) \ : \ 0 \le i < k\}.$$

The Petersen graph has $k = 5$ and $\pi(i) = 3i \,(\mathrm{mod}\ 5)$, for $i = 0, 1, 2, 3, 4$; GPGs have k odd and $\pi(s \cdot i \,(\mathrm{mod}\ k)) = i,\ i = 0, \ldots, k-1$, for a fixed integer s.

From Lemma 2.1 we know that $g(P_{k,\pi}) \ge |V(P_{k,\pi})| - 1 = 2k - 1$. We will show that for any k and π such that $P_{k,\pi}$ is 3-edge-colorable, we have the equality

$$g(P_{k,\pi}) = 2k - 1.$$

Each cubic GPG, other than the Petersen graph itself, is 3-edge-colorable [2]. Moreover, the class of 3-edge-colorable $P_{k,\pi}$'s includes the family of non-Hamiltonian GPGs with $k = 5 \,(\mathrm{mod}\ 6)$ and $s = 2$ (see [2] and references therein quoted). For an example of a 3-coloration of a GPG, see Figure 7.

The gossiping algorithm is described in Figure 8; it assumes that the edges of the graph are colored with the three colors 1, 2, and 3. It is easy to prove by induction on q that all the calls of **Phase** q, for $q \le (k-1)/2$, can actually be done. Therefore, after the first $(k-1)/2$ phases, each node a_i has the packet of $a_{i \pm j(\mathrm{mod}\ k)}$ for $j = 0, \ldots, (k-1)/2$; that is, it knows the packet of each other node in its own cycle; moreover, it knows the packet of $(k-1)/2$ nodes in the cycle on $\{b_0, \ldots, b_{k-1}\}$. Analogously, each b_i knows the packet of each other node in its own cycle and of $(k-1)/2$ nodes in $\{a_0, \ldots, a_{k-1}\}$.

Gossiping-3-color$(P_{k,\pi})$
Phase q $(1 \le q \le (k-1)/2)$ *[each phase consists of three communication rounds]*:
round t $(t = 1, 2, 3)$: make a call between the endpoints of each edge of color t.
Calls are made so that:
when an edge $(a_i, a_{i+1(\mathrm{mod}\ k)})$ is used, then a_i receives the packet of $a_{i+q(\mathrm{mod}\ k)}$, and $a_{i+1(\mathrm{mod}\ k)}$ receives the packet of $a_{i+1-q(\mathrm{mod}\ k)}$;
when an edge $(b_i, b_{i+1(\mathrm{mod}\ k)})$ is used, then b_i receives the packet of $b_{i+q(\mathrm{mod}\ k)}$, and $b_{i+1(\mathrm{mod}\ k)}$ receives the packet of $b_{i+1-q(\mathrm{mod}\ k)}$;
when the edge $(a_i, b_{\pi(i)})$ is used, then a_i receives the packet of some b_j, $0 \le j \le k-1$, and $b_{\pi(i)}$ receives the packet of some a_j, $0 \le j \le k-1$.
Phase $3(k-1)/2 + q$ $(1 \le q \le (k+1)/2)$ *[the phase consists of one communication round]*:
node a_i (resp., $b_{\pi(i)}$), for $i = 0, \ldots, k-1$, sends to $b_{\pi(i)}$ (resp., a_i) the packet of some a_j (resp., b_j) it has not already sent to it.

FIG. 8. *Gossiping algorithm in* $P_{k,\pi}$.

Therefore, the calls between nodes in $\{a_0, \ldots, a_{k-1}\}$ and in $\{b_0, \ldots, b_{k-1}\}$ of the last $(k+1)/2$ communication rounds allow completion of the knowledge of each node in the graph.

3. Gossiping by exchanging more than one packet at a time. In this section we shall study the minimum number of time units $g_{F_1}(p, G)$ necessary to perform gossiping in a graph G, under the restriction that at each time instant communicating nodes can exchange up to p packets, p fixed but arbitrary otherwise. We assume that p is smaller than the number of nodes of the graph G; otherwise, the problem is equivalent to the classical one. Again, for ease of notation, we shall write $g(p, G)$ to denote $g_{F_1}(p, G)$.

3.1. Lower bounds. First, we shall present a simple lower bound on $g(p, G)$ based on elementary counting arguments. Nonetheless, we shall prove in the sequel that the obtained lower bound is tight for complete graphs with an even number of nodes and for hypercubes. In order to derive the lower bound, let us define $I(p, t)$ as the maximum number of packets a vertex can have possibly received after t communication rounds in *any* graph. Since at each round i, with $1 \le i \le t$, any vertex can receive at most $\min\{p, 2^{i-1}\}$ packets, it follows that

$$I(p, t) = 1 + \sum_{i=1}^{t} \min\{p, 2^{i-1}\}, \tag{11}$$

or equivalently,

$$I(p, t) = 1 + \sum_{i=1}^{\lceil \log p \rceil} 2^{i-1} + p(t - \lceil \log p \rceil) = 2^{\lceil \log p \rceil} + p(t - \lceil \log p \rceil) \tag{12}$$

for any $t \ge \lceil \log p \rceil$. Therefore, for any graph $G = (V, E)$, the gossiping time $g(p, G)$ is always lower bounded by the smallest integer $t^\star$ for which $I(p, t^\star) \ge |V|$. Since $t^\star$

is obviously greater than or equal to $\lceil \log |V| \rceil \geq \lceil \log p \rceil$, we can use (12) and obtain

$$g(p,G) \geq \lceil \log p \rceil + \left\lceil \frac{1}{p}\left(|V| - 2^{\lceil \log p \rceil}\right) \right\rceil.$$

Moreover, notice that if the number of nodes in the graph is odd, then at each round there is a node that does not receive any message. This implies that after any round t there exists a node that can have possibly received at most $I(p,t-1)$ packets. Therefore,

$$g(p,G) \geq \lceil \log p \rceil + \left\lceil \frac{1}{p}\left(|V| - 2^{\lceil \log p \rceil}\right) \right\rceil + 1.$$

The above arguments give the following lemma.

LEMMA 3.1. *For any graph $G=(V,E)$, $|V|=n$, and integer p such that $2^{\lceil \log p \rceil} \leq n$, we have*

$$g(p,G) \geq \begin{cases} \lceil \log p \rceil + \left\lceil \frac{1}{p}\left(n - 2^{\lceil \log p \rceil}\right) \right\rceil & \text{if } n \text{ is even,} \\ \lceil \log p \rceil + \left\lceil \frac{1}{p}\left(n - 2^{\lceil \log p \rceil}\right) \right\rceil + 1 & \text{if } n \text{ is odd.} \end{cases}$$

Using similar arguments, we can also generalize the lower bound (1) that we established in section 2.1 for $p=1$ to general values of p.

LEMMA 3.2. *Let $G=(V,E)$ be a graph with n vertices and let $\mu(G)$ be the size of a maximum matching in G. For any integer p such that $2^{\lceil \log p \rceil} \leq n$,*

$$g(p,G) \geq \lceil \log p \rceil + \left\lceil \frac{1}{p}\left(\frac{n(n-1)}{2\mu(G)} - 2^{\lceil \log p \rceil} + 1\right) \right\rceil.$$

Remark 3.1. Given a gossiping algorithm $\mathcal{A}$ for a graph G that uses messages of size not larger than p, we can easily derive from it an algorithm $\mathcal{B}$ to gossip in G with messages of size $q < p$. Indeed, if a message of size $> q$ is sent during a call of $\mathcal{A}$, then we can split this call into more calls, each transmitting up to q packets. For example, we can use this observation to derive the following bound:

$$g(p,G) \leq \lfloor \log p \rfloor + 1 + 2(g(2p,G) - \lfloor \log p \rfloor - 1) = 2g(2p,G) - \lfloor \log p \rfloor - 1. \tag{13}$$

Bound (13) can be proved by noticing that during the first $\lfloor \log p \rfloor + 1$ calls of the algorithm attaining $g(2p,G)$, the exchanged messages necessarily have size less than or equal to p. From this observation and Theorem 2.5, it follows, for example, that for the complete bipartite graph $g(2,K_{r,s}) \geq (g(1,K_{r,s})+1)/2 = \lceil (r+s-1)r/(2s) \rceil + 1$; it is not difficult to derive an algorithm similar to the one in Figure 6 attaining the equality.

3.2. Rings and paths. Let $g(\infty,G)$ denote the gossiping time of the graph G in the absence of any restriction on the size of the messages. It is obvious that for each p, $g(p,G) \geq g(\infty,G)$ holds; it is possible to see that equality holds for any $p \geq 2$ when G is either the ring C_n or the path P_n on n nodes.

It is well known that [17]

$$g(\infty,P_n) = 2\left\lceil \frac{n}{2} \right\rceil - 1 \quad \text{and} \quad g(\infty,C_n) = \begin{cases} n/2 & \text{if } n \text{ is even,} \\ (n+3)/2 & \text{if } n \text{ is odd.} \end{cases}$$

We just point out that it is easy to see that the algorithms attaining $g(\infty, C_n)$ and $g(\infty, P_n)$ do not need to send more than two packets at a time. Therefore, the following results hold.

THEOREM 3.1. *For each $n \geq 3$ and $p \geq 2$,*

$$g(p, C_n) = g(2, C_n) = \begin{cases} n/2 & \text{if } n \text{ is even,} \\ (n+3)/2 & \text{if } n \text{ is odd.} \end{cases}$$

THEOREM 3.2. *For each $n \geq 2$ and $p \geq 2$,*

$$g(p, P_n) = g(2, P_n) = 2\left\lceil \frac{n}{2} \right\rceil - 1.$$

3.3. Complete graphs. In this section we study the gossiping time of the complete graph K_n on n nodes. We shall denote by $\{0, 1, \ldots, n-1\}$ the vertex set of K_n. We recall that $g(\infty, K_n)$ is equal to $\lceil \log n \rceil$ if n is even, and $\lceil \log n \rceil + 1$ if n is odd.

THEOREM 3.3. *For each even integer n and integer p such that $2^{\lceil \log p \rceil} \leq n$,*

$$g(p, K_n) = \lceil \log p \rceil + \left\lceil \frac{n - 2^{\lceil \log p \rceil}}{p} \right\rceil.$$

Proof. The lower bound follows from Lemma 3.1. We now give a gossiping algorithm for K_n that uses the optimal number of rounds. For each node v, with v even and $0 \leq v \leq n-1$, define the sequence of nodes v_t as

$$v_t = \begin{cases} v + 2^t - 1 \pmod n & \text{if } 1 \leq t \leq \lceil \log p \rceil, \\ v + 2^{\lceil \log p \rceil} - 1 + \tau p + 1 \pmod n & \text{if } t = \lceil \log p \rceil + \tau, \text{ with } \tau \geq 1 \text{ and } p \cdot \tau \text{ odd,} \\ v + 2^{\lceil \log p \rceil} - 1 + \tau p \pmod n & \text{if } t = \lceil \log p \rceil + \tau, \text{ with either } p \text{ or } \tau \geq 1 \text{ even.} \end{cases} \tag{14}$$

Note that for each t the set $M_t = \{(v, v_t) \ : \ v \text{ even}, 0 \leq v < n\}$ is a perfect matching between even and odd nodes. Finally, for each integer $\tau \geq 1$, for each even node v, with $0 \leq v \leq n-1$, define

$$P_{\text{even}}(v, \tau) = \begin{cases} \{v + i (\text{mod} n) : 1 \leq i \leq p\} & \text{if } p \text{ and } \tau \text{ are odd,} \\ \{v + i (\text{mod} n) : 0 \leq i \leq p-1\} & \text{otherwise,} \end{cases} \tag{15}$$

and for each odd node v, with $0 \leq v \leq n-1$,

$$P_{\text{odd}}(v, \tau) = \begin{cases} \{v - i (\text{mod} n) : 1 \leq i \leq p\} & \text{if } p \text{ and } \tau \text{ are odd,} \\ \{v - i (\text{mod} n) : 0 \leq i \leq p-1\} & \text{otherwise.} \end{cases} \tag{16}$$

Consider the gossiping algorithm given in Figure 9 and let $I_n(v, t)$ denote the set of nodes whose packets are known by v by the end of round t. For each node v the size of $I_n(v, t)$ doubles at each round of **Phase 1** and increases by p in every round of **Phase 2**. Indeed, it is immediate to see that for each $t = 1, \ldots, \lceil \log p \rceil$

$$I_n(v, t) = \begin{cases} \{v + i (\text{mod} n) : 0 \leq i \leq 2^t - 1\} & \text{if } v \text{ is even,} \\ \{v - i (\text{mod} n) : 0 \leq i \leq 2^t - 1\} & \text{if } v \text{ is odd,} \end{cases} \tag{17}$$

Gossiping-even(p, K_n)
Phase 1
Round t, $1 \le t \le \lceil \log p \rceil$: For each even node v
nodes v and v_t exchange all the packets they know.

Phase 2
Round $t = \lceil \log p \rceil + \tau$, $1 \le \tau \le \left\lceil \frac{n-2^{\lceil \log p \rceil}}{p} \right\rceil$: For each even node v
node v sends to v_t the packets of nodes in $P_{\text{even}}(v, \tau)$ and
node v_t sends to v the packets of nodes in $P_{\text{odd}}(v_t, \tau)$.

FIG. 9. *Gossiping algorithm in K_n, n even.*

and for each $\tau = 1, \ldots, \left\lceil \frac{n-2^{\lceil \log p \rceil}}{p} \right\rceil$

(18)
$$I_n(v, \lceil \log p \rceil + \tau) = \begin{cases} \{v + i (\mathrm{mod} n) : 0 \le i \le 2^{\lceil \log p \rceil} + \tau p - 1\} & \text{if } v \text{ is even,} \\ \{v - i (\mathrm{mod} n) : 0 \le i \le 2^{\lceil \log p \rceil} + \tau p - 1\} & \text{if } v \text{ is odd.} \end{cases}$$

Hence, $I_n\left(v, \lceil \log p \rceil + \left\lceil \frac{n-2^{\lceil \log p \rceil}}{p} \right\rceil\right) = \{0, \ldots, n-1\} = V$ for each node v. ∎

Remark 3.2. A close look at the algorithm **Gossiping-even**(p, K_n) reveals that the calls are always made between even and odd nodes. Therefore, the same protocol works in the complete bipartite graphs $K_{r,r}$ from which we get that for any p and r

$$g(p, K_{r,r}) = g(p, K_{2r}) = \lceil \log p \rceil + \left\lceil \frac{1}{p}\left(2r - 2^{\lceil \log p \rceil}\right) \right\rceil.$$

We now consider the case of complete graphs with odd number of nodes.

THEOREM 3.4. *For each odd integer N and integer p such that $2^{\lceil \log p \rceil} \le N + 1$,*

$$\lceil \log p \rceil + \left\lceil \frac{N - 2^{\lceil \log p \rceil}}{p} \right\rceil + 1 \le g(p, K_N) \le \lceil \log p \rceil + \left\lceil \frac{N + 1 - 2^{\lceil \log p \rceil}}{p} \right\rceil + 2.$$

Proof. The lower bound follows from Lemma 3.1. To prove the upper bound, we show that the algorithm **Gossiping-odd**(p, K_N) given in Figure 10 completes gossiping in K_N in $\lceil \log p \rceil + \left\lceil \frac{N+1-2^{\lceil \log p \rceil}}{p} \right\rceil + 2$ rounds. The algorithm **Gossiping-odd**(p, K_N) is described in terms of the algorithm **Gossiping-even**(p, K_n), where $n = N + 1$.

Let V_t, $P_{\text{even}}(v, \tau)$, and $P_{\text{odd}}(v, \tau)$ be defined as in (14), (15), and (16), respectively. In order to show the correctness of **Gossiping-odd**(p, K_N), let us first consider **Phase 1**. At round t, for $1 \le t \le \lceil \log p \rceil$, node $N + 1 - 2^t$ does not receive the information of the nodes in $I_n(N, t) - \{N\}$. It is easy to see that the nodes that *do not have* the packets of *all* the nodes in $I_n(v, t)$ are the nodes in the set X_t defined by $X_1 = \emptyset$, and

$$\begin{aligned} X_t = X_{t-1} \cup \{ & v + 2^t - 1 \ (\mathrm{mod}\ n) \ : \ v \in X_{t-1} \text{ even } \} \\ & \cup \{ v - 2^t + 1 \ (\mathrm{mod}\ n) \ : \ v \in X_{t-1} \text{ odd } \} \cup \{N + 1 - 2^t\} \end{aligned}$$

Gossiping-odd(p, K_N)
Phase 1

Round t, $1 \le t \le \lceil \log p \rceil$: For each even node v, with $v \ne N+1-2^t$, nodes v and v_t exchange all the packets they know;
Round $t = \lceil \log p \rceil + 1$: each node v with $v \in \{3+4i \ : \ 0 \le i \le 2^{\lceil \log p \rceil - 2} - 2\} \cup \{N-3-4i \ : \ 0 \le i \le 2^{\lceil \log p \rceil - 2} - 1\}$ receives from $v+2$ a message containing the packets of all the nodes in $\{N - 2^{\lceil \log p \rceil - 1} + 1, \ldots, N-1\}$.

Phase 2

Round $t = \lceil \log p \rceil + 1 + \tau$, $1 \le \tau \le \left\lceil (N+1-2^{\lceil \log p \rceil})/p \right\rceil$: For each even v with $v_{t-1} \ne N$ node v sends to v_{t-1} the packets of nodes in $P_{\text{even}}(v, \tau)$ and v_{t-1} sends to v the packets of nodes in $P_{\text{odd}}(v_{t-1}, \tau)$.
Round $t = \lceil \log p \rceil + \left\lceil (N+1-2^{\lceil \log p \rceil})/p \right\rceil + 2$: Each node v such that $v_{t-1} = n-1$ for some $t = \lceil \log p \rceil + 1 + \tau$ with $1 \le \tau \le \left\lceil (N+1-2^{\lceil \log p \rceil})/p \right\rceil + 1$ receives from $v+1$ a message containing the packets of the nodes in $P_{\text{odd}}(N, \tau)$.

FIG. 10. *Gossiping algorithm in K_N, N odd.*

for $2 \le t \le \lceil \log p \rceil$. This gives

$$X_t = \{3+4i \ : \ 0 \le i \le 2^{t-2} - 2\} \cup \{N-3-4i \ : \ 0 \le i \le 2^{t-2} - 1\} \tag{19}$$

for $t = 2, \ldots, \lceil \log p \rceil$.

Moreover, each node in X_t has at least those packets of all nodes in $I(v,t) - I(N, t-1)$. Therefore, at the end of round $\lceil \log p \rceil$, each node in $X_{\lceil \log p \rceil}$ lacks at most the packets of the nodes in $I(N, \lceil \log p \rceil - 1) = \{N - 2^{\lceil \log p \rceil - 1} + 1, N - 2^{\lceil \log p \rceil - 1} + 2, \ldots, N-1\}$, and the calls of round $\lceil \log p \rceil + 1$ between each node $v \in X_{\lceil \log p \rceil}$ and $v+2 \notin X_{\lceil \log p \rceil}$ assure that each node knows the packets of all nodes in $I(v, \lceil \log p \rceil)$.

Now consider **Phase 2**. It is immediate that at round t each node receives p new packets, except for the even node v such that $v_{t-1} = N$. Hence, after the calls of round $\lceil \log p \rceil + \left\lceil \frac{n - 2^{\lceil \log p \rceil}}{p} \right\rceil + 2$, each node knows the packet of each of the other $N-1$ nodes. □

For N odd, we believe that the true value of $g(p, K_N)$ is $\lceil \log p \rceil + \left\lceil \frac{N - 2^{\lceil \log p \rceil}}{p} \right\rceil + 1$; we can verify this equality for small values of N and p. In case $p = 2$, Theorem 3.1 and Lemma 3.1 tell us that $g(2, K_N) = (N+3)/2 = g(2, C_N)$, for each odd $N \ge 2$. Moreover, we can prove the following theorem.

THEOREM 3.5. *If p is a multiple of 4 then $g(p, K_N) = \lceil \log p \rceil + \left\lceil \frac{N - 2^{\lceil \log p \rceil}}{p} \right\rceil + 1$.*

Proof. Execute the first $\lceil \log p \rceil$ rounds of **Gossiping-odd**(p, K_N): from (19) we know that the nodes that have not received the packets of all nodes in $I_n(v, \lceil \log p \rceil)$ are those in the set

$$X_{\lceil \log p \rceil} = \{3+4i \ : \ 0 \le i \le 2^{\lceil \log p \rceil - 2} - 2\} \cup \{N-3-4i \ : \ 0 \le i \le 2^{\lceil \log p \rceil - 2} - 1\}. \tag{20}$$

Continue the gossiping process as follows.

Round $t = \lceil \log p \rceil + \tau$, $1 \le \tau \le \left\lceil (N - 2^{\lceil \log p \rceil})/p \right\rceil - 1$: For each even v with $v_t \ne N$, v sends to v_t the packets of nodes in $P_{\text{even}}(v, \tau)$ and v_t sends to v the packets of nodes in $P_{\text{odd}}(v_t, \tau)$.

The set $X_{\lceil \log p \rceil + \tau}$ of the nodes that at round $\lceil \log p \rceil + \tau$ do not have the packets of all nodes in $I_n(v, \lceil \log p \rceil + \tau)$ satisfies

$$X_{\lceil \log p \rceil + \tau} = X_{\lceil \log p \rceil + \tau - 1} \cup \{v \ : \ v_t \in X_{\lceil \log p \rceil} \cup \{N\}\}.$$

We can then deduce that for each $\tau \leq \left\lceil (N - 2^{\lceil \log p \rceil})/p \right\rceil - 1$,

$$\begin{aligned} X_{\lceil \log p \rceil + \tau} = &\{3 + 4i \ : \ 0 \leq i \leq 2^{\lceil \log p \rceil - 2} + \tau p/4 - 1\} \\ &\cup \{N - 3 - 4i \ : \ 0 \leq i \leq 2^{\lceil \log p \rceil - 2} + \tau p/4 - 2\}. \end{aligned}$$

Now consider the matchings $M = \{(v, v+1) \ : \ v \text{ is even}\}$ and $M' = \{(v, v+3) \ : \ v \text{ is even}\}$; it is easy to see that gossiping can be completed in two more rounds by exchanging calls during rounds $\lceil \log p \rceil + \left\lceil \frac{N - 2^{\lceil \log p \rceil}}{p} \right\rceil$ and $\lceil \log p \rceil + \left\lceil \frac{N - 2^{\lceil \log p \rceil}}{p} \right\rceil + 1$ along the edges of M and M' if $N = 1 (\mathrm{mod}\ 4)$ or the edges of M' and M if $N = 3 (\mathrm{mod}\, 4)$, respectively. □

3.4. Hypercube. In the next theorem we shall determine $g(p, G)$ for any p when the graph G is the d-dimensional hypercube H_d with 2^d nodes.

THEOREM 3.6. *For each integer* $p < 2^d$, $\quad g(p, H_d) = \lceil \log p \rceil + \left\lceil \frac{1}{p}\left(2^d - 2^{\lceil \log p \rceil}\right) \right\rceil$.

Proof. The lower bound follows from Lemma 3.1. We now prove the matching upper bound. Let p be fixed. Denote by t_d the minimum integer such that $I(p, t_d) \geq 2^d$, where $I(p, t_d)$ is given in (11). We shall show that there exists a gossiping protocol that requires t_d rounds. Notice that $t_d = \lceil \log p \rceil + \left\lceil \frac{1}{p}\left(2^d - 2^{\lceil \log p \rceil}\right) \right\rceil$.

The proof is by induction on d. The assertion is trivially true for $d = 1$; suppose that there exists a gossiping protocol in H_d that takes t_d rounds to be completed and that satisfies the additional property that after any round $t \leq t_d - 1$ each vertex knows exactly $I(p, t)$ packets. We shall exhibit a gossiping protocol in H_{d+1} that takes t_{d+1} rounds to be completed and that also satisfies the aforesaid additional property.

Case 1: $I(p, t_d) = 2^d$. This implies that in the last round of the gossiping protocol in H_d—the t_dth—each vertex must receive exactly $\min\{p, 2^{d-1}\}$ packets. Consider the following protocol in the $(d+1)$-dimensional hypercube H_{d+1}: split H_{d+1} into two hypercubes of dimension d according to the value of its $(d+1)$th dimension; during the first t_d rounds gossip separately in each d-dimensional subcube according to the protocol whose existence is guaranteed by the induction hypothesis. After t_d rounds each vertex has received all the information of the subcube it belongs to; i.e., according to the hypothesis of this case, each vertex has received exactly $I(p, t_d) = 2^d$ packets. Now, in successive rounds, exchange packets along dimension $d+1$ in H_{d+1} by sending either all 2^d packets in one round, if $p > 2^d$, or p packets per round except in the last one, where one sends $2^d - p\lfloor 2^d/p \rfloor$ (if nonzero) packets. It is clear that this protocol requires t_{d+1} rounds to be completed. Moreover, for each t, with $0 \leq t \leq \lfloor 2^d/p \rfloor$, after round $t_d + t \leq t_{d+1} - 1$ each node in H_{d+1} knows exactly $I(p, t_d) + pt = I(p, t_d + t)$ packets. Hence the protocol for H_{d+1} satisfies the inductive hypothesis.

Case 2: $I(p, t_d) > 2^d$. This implies that $p < 2^{d-1}$; otherwise it is easy to check that one would have $t_d = d$ and $I(p, t_d) = 1 + \sum_i 2^{i-1} = 2^d$. Consider the protocol in H_d whose existence is implied by the induction hypothesis. By inductive hypothesis at round $t_d - 1$ each vertex has received $I(p, t_d - 1)$ packets, and in the last round, receives α packets, with $\alpha < p$; otherwise, we

would be again in Case 1.

Let $\mathcal{M} = \cup_{i=1}^{2^{d-1}}(x_i, y_i)$ be the perfect matching used in the last round, i.e., the round t_d, of the protocol on H_d, and let A_i (resp., B_i) be the set of new packets that x_i (resp., y_i) receives in this last round. Note that $A_i \cap B_i = \emptyset$ and $|A_i| = |B_i| = \alpha$. For what follows, let C_i and D_i be two sets of packets such that $|C_i| = |D_i| = p - \alpha$ and $C_i \cap A_i = \emptyset$, $D_i \cap A_i = \emptyset$, $C_i \cap B_i = \emptyset$, $D_i \cap B_i = \emptyset$, and $C_i \cap D_i = \emptyset$. Such sets exist since $|A_i| + |B_i| + |C_i| + |D_i| = 2p < 2^d$. Consider now the following gossiping protocol in H_{d+1}. Split H_{d+1} according to the value of the $(d+1)$th dimension in two subcubes H_d and H'_d of dimension d; during the first $t_d - 1$ rounds, gossip in H_d and H'_d separately. At the end of this phase each vertex knows $2^d - \alpha$ packets. Now, for each node x in H_d, denote by x' its neighbor in H'_d. In the next round, exchange p packets along dimension $d+1$ in such a way that x_i (resp., y_i, x'_i, y'_i) sends to x'_i (resp., y'_i, x_i, y_i) p packets including C_i (resp., D_i, C'_i, D'_i) and not D_i (resp., C_i, D'_i, C'_i).

In the next round, exchange p packets along the matching $\mathcal{M}$ in such a way that x_i (resp., y_i) sends to y_i (resp., x_i) packets in $B_i \cup C'_i$ (resp., $A_i \cup D'_i$), and x'_i (resp., y'_i) sends to y'_i (resp., x'_i) all packets in $B'_i \cup C_i$ (resp., $A'_i \cup D_i$). After the above $t_d + 1$ rounds we are sure that each vertex x_i (resp., x'_i) knows all the packets of the subcube it belongs to, and so we can finish the protocol by sending packets along dimension $d+1$ in such a way that p new packets are received during each round (except possibly the last final round). Therefore, for each t, with $1 \le t \le 1 + \lfloor 2^d/p \rfloor$, each node in H_{d+1} after round $t_d + t - 1 \le t_{d+1} - 1$ knows exactly $I(p, t_d - 1) + p\,t = I(p, t_d + t - 1)$ packets. Hence this protocol in H_{d+1} satisfies all the induction hypothesis. □

Remark 3.3. It is worth pointing out that the obvious inequality

$$g_{H_1}(p, G) \le 2 g_{F_1}(p, G) \tag{21}$$

and the above theorem allow us to improve the upper bound on $g_{H_1}(p, H_d)$ given by Theorem 4 of [4] for all values of p which are not powers of two. Indeed, the authors of [4] have $g_{H_1}(p, H_d) \le 2d + 2^{d+1}/p - 2/p$, while from Theorem 3.6 and (21), we get Theorem 3.7.

THEOREM 3.7. *For each integer* $p < 2^d$,

$$g_{H_1}(p, H_d) \le 2\lceil \log p \rceil + 2 \left\lceil \frac{1}{p}\left(2^d - 2^{\lceil \log p \rceil}\right) \right\rceil.$$

4. p-optimal graphs. In this section we consider the problem of estimating the minimum possible number of edges in any graph in which gossiping can be performed in the minimum possible number of rounds. We consider only networks with an even number of nodes. More formally, for any even integer n and integer p such that $2^{\lceil \log p \rceil} \le n$, let us denote by $g(p, n)$ the minimum gossiping time of any graph with n nodes, that is (cf. Theorem 3.3),

$$g(p, n) := \min_{G\,:\,|V(G)|=n} g(p, G) = \lceil \log p \rceil + \left\lceil \frac{n - 2^{\lceil \log p \rceil}}{p} \right\rceil,$$

and by $\mathcal{M}(p, n)$, the quantity

$$\mathcal{M}(p,n) := \min\{m \ : \text{ there exists } G=(V,E) \text{ with } |V|=n,\ |E|=m,\ g(p,G)=g(p,n)\}.$$

Our objective is to find significant bounds on the function $\mathcal{M}(p,n)$. From a practical point of view, an interconnection network G having gossiping time $g(p,G)=g(p,n)$ and $\mathcal{M}(p,n)$ edges represents the most economical network, if our main concern is the number of communication lines, that still preserves the communication capabilities of the complete graph, as far as gossiping is concerned. The analogous problem of estimating the minimum possible number of edges in a network in which broadcasting can be performed in minimum time has been extensively studied (see [5, 13] and references therein quoted). Estimating $\mathcal{M}(p,n)$ seems a much harder task. Even when p is unbounded, only few results are known [21].

DEFINITION 4.1. *Given a graph $G(V,E)$ on n nodes and an integer p such that $2^{\lceil \log p \rceil} \leq n$ we say that G is p-optimal if $g(p,G)=g(p,n)$ and $|E|=\mathcal{M}(p,n)$, that is, if G is a sparsest graph among all the graphs with n nodes and minimum gossiping time $g(p,n)$.*

We first consider the special cases $p=1$ and $p=2$ that admit a very simple solution, and afterwards we consider the general case, that is, $p \geq 3$.

4.1. Sending $p \leq 2$ items per round. We have shown in sections 2.2 and 3.2 that for the ring C_n on n nodes

$$g(1,C_n)=g(1,n)=2\left\lceil \frac{n}{2} \right\rceil - 1 \text{ and } g(2,C_n)=g(2,n)=\begin{cases} n/2 & \text{if } n \text{ is even,} \\ (n+3)/2 & \text{if } n \text{ is odd.} \end{cases}$$

Consider any connected graph (tree) G with n nodes and $m \leq n-1$ edges. The lower bound given in Corollary 2.2 tells us that $g(1,G) \geq 2n-3$. Moreover, it is easy to verify that the inequality $g(1,G) \leq 2g(2,G)-1$ holds. The above two inequalities imply that $g(1,G) > g(1,n) = n-1$ for each $n \geq 3$ and $g(2,G) \geq (g(1,G)+1)/2 \geq (2n-2)/2 = n-1 \geq g(2,n)$ for each $n \geq 2$ with $n \neq 3$. It is easy to see that P_3 is also optimal for $p=2$. We have then proved the following theorem.

THEOREM 4.1. *$\mathcal{M}(1,2)=\mathcal{M}(2,2)=1$, $\mathcal{M}(1,3)=3$, $\mathcal{M}(2,3)=2$, and for each $n \geq 4$*

$$\mathcal{M}(1,n)=\mathcal{M}(2,n)=n.$$

4.2. Sending $p \geq 3$ items per round. In this section we study p-optimal graphs for $p \geq 3$. We recall that such graphs are to be sought among those graphs having gossiping time equal to $g(p,n)$. Let us first recall that for each $p \geq 3$ the ring C_n is not p-optimal; indeed, from the results of section 3.2 we have $\min_q g(q,C_n) = g(2,C_n) > g(p,n)$.

We have proved in section 3.4 that the hypercube H_d has minimum gossiping time for each value of p; moreover, it was shown in [21] that H_d is p-optimal for each $p \geq 2^{d-1}$ (equivalently, for p unbounded); that is, H_d has the minimum number of edges among all the networks with gossiping time $g(\infty,2^d)=d$. A natural question is whether the hypercube is p-optimal for other values of $p < 2^{d-1}$. The results of section 4.3 will imply a negative answer to the above question.

Let $d(p,n)$ be the minimum possible degree a node can have in any p-optimal graph on n nodes.

THEOREM 4.2. $d(p,n) \geq \lfloor \log p \rfloor + 1 - \left\lfloor \log \left(\left\lceil \frac{n-2^{\lceil \log p \rceil}}{p} \right\rceil p - n + 2^{\lceil \log p \rceil} + 1 \right) \right\rfloor$.

Proof. Denote by $r(p,t)$ the maximum number of items a node can receive with a call made at round t, and by $I(p,t) = 1 + \sum_{i=1}^{t} r(p,i)$, the maximum possible number of items a node can have received by round t. We recall that $I(p,0) = r(p,1) = 1$ and

$$r(p,t) = \max\{2^{t-1}, p\}$$

and

$$I(p,t) = \begin{cases} 2^t & \text{if } t \leq \lfloor \log p \rfloor + 1, \\ 2^{\lfloor \log p \rfloor + 1} + (t - \lfloor \log p \rfloor - 1)p & \text{if } t > \lfloor \log p \rfloor + 1. \end{cases}$$

Fix any gossiping protocol $\mathcal{P}$ that completes in $g(p,n)$ rounds. We denote by $r(p,t,v)$ the number of items node v receives at round t of $\mathcal{P}$ and let

$$I(p,t,v) = 1 + \sum_{i=1}^{t} r(p,i,v);$$

obviously $r(p,t,v) \leq r(p,t)$ and $I(p,t,v) \leq I(p,t)$ for each $t = 1, \ldots, g(p,n)$.

In order to prove the desired lower bound on $d(p,n)$ we show that any node has to make calls with at least $\lfloor \log p \rfloor + 1 - \left\lfloor \log \left(\left\lceil \frac{n-2^{\lceil \log p \rceil}}{p} \right\rceil p - n + 2^{\lceil \log p \rceil} + 1 \right) \right\rfloor$ *different* neighbors during the first $\lfloor \log p \rfloor + 1$ rounds of the protocol $\mathcal{P}$.

Fix a node v and suppose that v communicates with $\lfloor \log p \rfloor + 1 - \ell$ different neighbors during the first $\lfloor \log p \rfloor + 1$ rounds of $\mathcal{P}$. This means that there exist ℓ rounds, say $\tau_1, \ldots, \tau_\ell$, such that for each $i = 1, \ldots, \ell$ at round τ_i, v is either idle or makes a call with a node that will communicate again with v at some round δ_i with $\tau_i < \delta_i \leq \lfloor \log p \rfloor + 1$; we can bound $r(p, \tau_i, v)$ as follows.

i) If v does not participate in any call at round τ_i, then $r(p, \tau_i, v) = 0$.

ii) If v makes calls with a particular node, say w, at both rounds τ_i and δ_i, then at time δ_i node v will not receive again what it received at time τ_i from w, nor will it receive what it sent to w at time τ_i. Therefore, $r(p, \delta_i, v) \leq I(p, \delta_i - 1, w) - r(p, \tau_i, v) - r(p, \tau_i, w)$; that is,

$$r(p, \tau_i, v) + r(p, \delta_i, v) \leq I(p, \delta_i - 1, w) \leq 2^{\delta_i - 1} = r(p, \delta_i).$$

By i) and ii) we get that for each round $t \geq \lfloor \log p \rfloor + 1$,

$$\begin{aligned} I(p,t,v) = 1 + \sum_{i-1}^{t} r(p,i,v) &\leq I(p,t) - \sum_{i=1}^{\ell} 2^{\tau_i - 1} \\ &= 2^{\lfloor \log p \rfloor + 1} + (t - \lfloor \log p \rfloor - 1)p - \sum_{i=1}^{\ell} 2^{\tau_i - 1}. \end{aligned}$$

Recalling that n is even and the graph has minimum gossiping time $g(p,n) = \lceil \log p \rceil + \left\lceil \frac{n-2^{\lceil \log p \rceil}}{p} \right\rceil = \lfloor \log p \rfloor + 1 + \left\lceil \frac{n-2^{\lfloor \log p \rfloor + 1}}{p} \right\rceil$, the following inequality must be satisfied:

$$n \leq I(p, g(p,n)) - \sum_{i=1}^{\ell} 2^{\tau_i - 1} = 2^{\lfloor \log p \rfloor + 1} - \sum_{i=1}^{\ell} 2^{\tau_i - 1} + \left\lceil \frac{n - 2^{\lfloor \log p \rfloor + 1}}{p} \right\rceil p.$$

Noticing that $\sum_{i=1}^{\ell} 2^{\tau_i - 1} \leq \sum_{i=1}^{\ell} 2^{i-1} = 2^\ell - 1$, we get

$$\ell \leq \left\lfloor \log \left(\left\lceil \frac{n - 2^{\lfloor \log p \rfloor + 1}}{p} \right\rceil p - n + 2^{\lfloor \log p \rfloor + 1} + 1 \right) \right\rfloor$$

and the desired bound on $d(n,p)$ follows. □

4.3. A family of graphs with $O(\frac{n}{2} \log p)$ edges. In section 3.3 we have proved that $g(p, K_n) = g(p,n)$ for any even n and any p. Moreover, it is easy to see that in order to implement the gossiping protocol of Figure 9, only $O\left(n\left(\frac{n}{p} + \log p\right)\right)$ edges of K_n are needed. This implies that $\mathcal{M}(p,n) = O\left(n\left(\frac{n}{p} + \log p\right)\right)$. Actually, we can prove a much better bound. We will construct for any p and even n a graph $G_{p,n}$ with n nodes, $n(\lceil \log p \rceil + 1)/2$ edges, and optimal gossiping time $g(p, G_{p,n}) = g(p,n)$.

Let p be an even integer and define the sequence of integers $\mathbf{s}_p$ as follows: $\mathbf{s}_2 = (-1, 1)$ and for each $p = 2^m + q$ with $q \leq 2^m$, if $\mathbf{s}_{2^m} = (s_1, \ldots, s_{m+1})$, then

$$\mathbf{s}_p = (s_1, \ldots, s_{m+1}, s_{m+2}) \quad \text{with } s_{m+2} = \begin{cases} p + s_{m+1} & \text{if } m \text{ is even,} \\ -(p - s_{m+1}) & \text{if } m \text{ is odd.} \end{cases}$$

If p is odd define $\mathbf{s}_p = \mathbf{s}_{p+1}$.

Example 4.1. $\mathbf{s}_2 = (-1,1)$, $\mathbf{s}_3 = \mathbf{s}_4 = (-1,1,-3)$, $\mathbf{s}_5 = \mathbf{s}_6 = (-1,1,-3,3)$, $\mathbf{s}_7 = \mathbf{s}_8 = (-1,1,-3,5)$, $\mathbf{s}_9 = \mathbf{s}_{10} = (-1,1,-3,5,-5)$, $\mathbf{s}_{11} = \mathbf{s}_{12} = (-1,1,-3,5,-7)$. Let the node set be $V_n = \{0, 1, \ldots, n-1\}$. All operations on nodes will be performed modulo n. Define the matching

$$M_{p,n}(t) = \{(v, v + s_t) \mid v \in V_n \text{ is odd}\} \quad \text{for } t = 1 \ldots, \lceil \log p \rceil + 1,$$

and the graph $G_{p,n} = (V_n, E_{p,n})$ with $E_{p,n} = \cup_{t=1}^{\lceil \log p \rceil + 1} M_{p,n}(t)$; Figure 11 shows $G_{6,14}$.

One can check that at the end of the algorithm **Gossiping**$(G_{p,n})$, given in Figure 12, any node knows the packets of all the other nodes in $G_{p,n}$. Therefore, using Theorem 4.2, we get Theorem 4.3.

THEOREM 4.3. *For each integer p and even integer $n \geq 2^{\lceil \log p \rceil}$*

$$\tfrac{n}{2}\left(\lfloor \log p \rfloor + 1 - \left\lfloor \log \left(\left\lceil \tfrac{n - 2^{\lceil \log p \rceil}}{p} \right\rceil p - n + 2^{\lceil \log p \rceil} + 1 \right) \right\rfloor\right) \leq \mathcal{M}(p,n) \leq \tfrac{n}{2}(\lceil \log p \rceil + 1).$$

COROLLARY 4.1. *For each $p \geq 2$ and even integer n such that $n - 2^{\lceil \log p \rceil}$ is a multiple of p,*

$$\frac{n}{2}(\lfloor \log p \rfloor + 1) \leq \mathcal{M}(p,n) \leq \frac{n}{2}(\lceil \log p \rceil + 1).$$

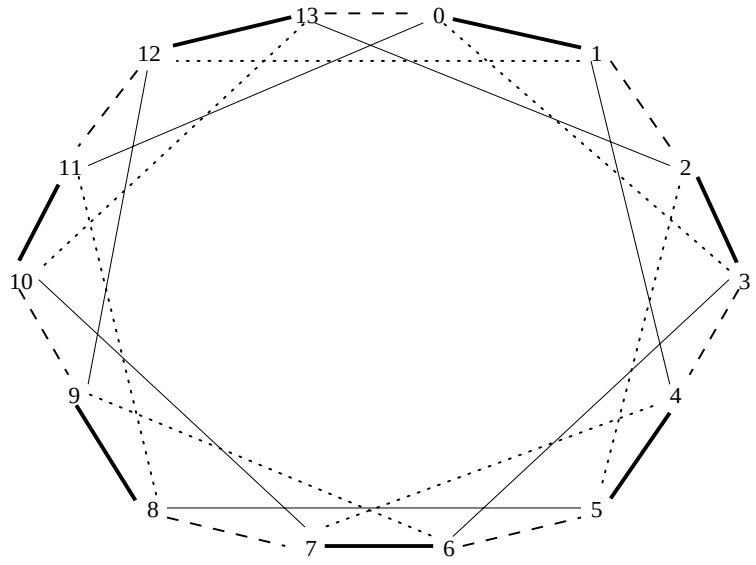
COROLLARY 4.2. *For each integer $q \geq 1$ and integer $r \geq 2$,*

$$\mathcal{M}(2^q, r2^q) = r2^{q-1}(q+1).$$

It is possible to improve the lower bound given in Theorem 4.2 proving that $d(2^q - 1, n) \geq q + 1 - \lfloor \log (\lceil (n-1)/(2^q - 1) \rceil (2^q - 1) - n + 2) \rfloor$, which together with Theorem 4.3 implies the following corollary.

COROLLARY 4.3. *For each integer $q \geq 1$ and odd integer r,*

$$\mathcal{M}(2^q - 1, r(2^q - 1) + 1) = (r(2^q - 1) + 1)(q+1)/2.$$



edge in the matching $M_{6,14}(1)$

edge in the matching $M_{6,14}(2)$

edge in the matching $M_{6,14}(3)$

edge in the matching $M_{6,14}(4)$

FIG. 11.

Gossiping($G_{p,n}$)

Let $\mathbf{s}_p = (s_1, \ldots, s_{\lceil \log p \rceil + 1})$;
At round t, $t = 1, \ldots, \lceil \log p \rceil + 1$, each node sends $\max\{2^{t-1}, p\}$ new items to its neighbor in $M_{p,n}(t)$;
At round $t = \lceil \log p \rceil + 1 + \tau$, $\tau = 1, \ldots, g(p,n) - \lceil \log p \rceil - 1$, consider the matching

$$M_{p,n}(t) = \begin{cases} M_{p,n}(\lceil \log p \rceil) & \text{if } \tau \text{ is odd,} \\ M_{p,n}(\lceil \log p \rceil + 1) & \text{if } \tau \text{ is even;} \end{cases}$$

then each node sends p new items (or fewer than p in the last round) to its neighbor in $M_{p,n}(t)$.

FIG. 12. *Gossiping algorithm in* $G_{p,n}$.

5. Concluding remarks and open problems. We have considered the problem of gossiping in communication networks under the restriction that communicating nodes can exchange up to a fixed number p of packets at each round. In the extremal case $p = 1$ we have given optimal algorithms to perform gossiping in several classes of graphs, including Hamiltonian graphs, paths, complete k-ary trees, and complete bipartite graphs. For arbitrary graphs we gave asymptotically matching upper and lower bounds.

In the case of arbitrary p we have determined the optimal number of communication rounds to perform gossiping under this hypothesis for complete graphs, hypercubes, rings, paths, and complete bipartite graphs $K_{r,r}$.

Several open problems remain in the area. We list the most important of them here.

• It would be interesting to determine the computational complexity of computing $g_{F_1}(1, G)$ ($g_{F_1}(p, G)$) for general graphs; it is very likely that it is NP-hard. (We know that computing $g_{F_1}(\infty, G)$ is NP-hard; see [20].)

• We have left open the problem of determining the gossiping time $g_{F_1}(1, G_{t,s})$, and more generally $g_{F_1}(p, G_{t,s})$, of rectangular grids $G_{t,s}$ with both t and s odd. We know from Corollary 2.1 that $g_{F_1}(1, G_{t,s}) \geq st + 1$. Does equality hold? We can prove that $g_{F_1}(1, G_{3,3}) = 10$. A general upper bound on $g_{F_1}(1, G_{t,s})$ can be obtained by observing that $G_{t,s} = P_t \times P_s$, where P_t and P_s are the paths on t and s nodes, respectively, and $\times$ denotes the cartesian graph product. Now, given two graphs $G = (V, E)$ and $H = (W, F)$, it is easy to see that $g_{F_1}(1, G \times H) \leq \min\{g_{F_1}(1, G) + |V| g_{F_1}(1, H),\ g_{F_1}(1, H) + |W| g_{F_1}(1, G)\}$, which, together with Corollary 2.5, immediately gives $g_{F_1}(1, G_{t,s}) \leq 2ts - 3 - \max\{t, s\}$.

• We know from (4) that for any graph G with n vertices one has $g_{F_1}(1, G) \geq n$ if n is odd, $g_{F_1}(1, G) \geq n - 1$ if n is even, and from Theorem 2.1, we get that the equality holds for Hamiltonian graphs. It would be interesting to characterize the class of graphs for which this lower bound is tight. We know from the results of section 2.5 that this class is larger than the class of the Hamiltonian graphs.

• In view of the possible NP-hardness of computing $g(p, G)$ for arbitrary graphs, it would be interesting to design efficient algorithms to compute gossiping protocols that complete in time "close" to $g(p, G)$. Such algorithms have recently been provided for $g(\infty, G)$ (see [14, 23]). However, the techniques used there do not seem to apply to the case of bounded p.

• We have given fairly tight bounds on the function $\mathcal{M}(p, n)$. It would be interesting to study the analogous quantity $\mathcal{M}_c(p, n)$ equal to the minimum number of edges in any graph in which gossiping can be performed in quasi-optimal time $g(p, n) + c$, where c is a small constant. In particular, we ask whether $\mathcal{M}_1(p, n) = O(n)$.

Acknowledgments. The authors wish to thank R. Ravi for the proof of Theorem 2.3. They also want to thank Levon Khachatrian and Jonny Bond for interesting discussions. The first author would like to thank the Dipartimento di Informatica ed Applicazioni, Università di Salerno, Italy, where part of this research was conducted, for inviting him.

REFERENCES

[1] N. Alon, F.R.K. Chung, and R.L. Graham, *Routing permutations on graphs via matchings*, in Proc. 25th ACM Symposium on the Theory of Computing (STOC '93), San Diego, CA, 1993, pp. 583–591.

[2] B. Alspach, *The classification of Hamiltonian generalized Petersen graphs*, J. Combin. Theory Ser. B, 34 (1983), pp. 293–312.
[3] A. Bagchi, E.F. Schmeichel, and S.L. Hakimi, *Sequential information dissemination by packets*, Networks, 22 (1992), pp. 317–333.
[4] A. Bagchi, E.F. Schmeichel, and S.L. Hakimi, *Parallel information dissemination by packets*, SIAM J. Comput., 23 (1994), pp. 355–372.
[5] J.-C. Bermond, P. Fraignaud, and J. Peters, *Antepenultimate broadcasting*, Networks, 25 (1995), pp. 125–137.
[6] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Prentice-Hall, Englewood Cliffs, NJ, 1989.
[7] B. S. Chlebus, K. Diks, and A. Pelc, *Optimal gossiping with short unreliable messages*, Disc. Appl. Math., 53 (1994), pp. 15–24.
[8] K. Diks and A. Pelc, *Efficient gossiping by packets in networks with random faults*, SIAM J. Disc. Math., 9 (1996), pp. 7–18.
[9] G. Fox, M. Johnsson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors, Volume* I, Prentice-Hall, Englewood Cliffs, NJ, 1988.
[10] P. Fraignaud and E. Lazard, *Methods and problems of communication in usual networks*, Disc. Appl. Math., 53 (1994), pp. 79–134.
[11] S. Fujita, *Gossiping in mesh-bus computers by packets with bounded length*, IPS Japan SIGAL, 36 (1993), pp. 41–48.
[12] M. Fürer and B. Raghavachari, *Approximating the minimum degree spanning tree to within one from the optimal degree*, in Proc. Third Annual ACM–SIAM Symposium on Discrete Algorithms (SODA '92), Orlando, FL, 1992, pp. 317–324.
[13] M. Grigni and D. Peleg, *Tight bounds on minimum broadcast networks,* SIAM J. Disc. Math., 5 (1992), pp. 207–222.
[14] G. Kortsarz and D. Peleg, *Approximation algorithms for minimum time broadcast*, SIAM J. Disc. Math., 8 (1995), pp. 401–427.
[15] S. M. Hedetniemi, S. T. Hedetniemi, and A. Liestman, *A survey of gossiping and broadcasting in communication networks*, Networks, 18 (1988), pp. 129–134.
[16] A. Hily and D. Sotteau, *Communications in Bus Networks*, in Parallel and Distributed Computing, Lectures Notes in Computer Science 805, M. Cosnard, A. Ferreira, and J. Peters, eds., Springer-Verlag, New York, 1994, pp. 197–206.
[17] J. Hromkovič, R. Klasing, B. Monien, and R. Peine, *Dissemination of Information in Interconnection Networks* (*Broadcasting and Gossiping*), in Combinatorial Network Theory, F. Hsu and D.-Z. Du, eds., Kluwer Academic Publishers, Norwell, MA, 1995, pp. 125–212.
[18] S. L. Johnsson and C. T. Ho, *Matrix multiplication on Boolean cubes using generic communication primitives*, in Parallel Processing and Medium-Scale Multiprocessors, A. Wouk, ed., SIAM, Philadelphia, 1989, pp. 108–156.
[19] D. W. Krumme, *Fast gossiping for the hypercube*, SIAM J. Comput., 21 (1992), pp. 365–380.
[20] D. W. Krumme, K. N. Venkataraman, and G. Cybenko, *Gossiping in minimal time*, SIAM J. Comput., 21 (1992), pp. 111–139.
[21] R. Labhan, *Some minimum gossip graphs*, Networks, 23 (1993), pp. 333–341.
[22] R. Ravi, private communication.
[23] R. Ravi, *Rapid rumor ramification: Approximating the minimum broadcasting time*, in Proc. 35th Annual Symposium on Foundations of Computer Science (FOCS '94), 1994, pp. 202–213.
[24] J. de Rumeur, *Communication dans les Reseaux de Processeur*, Masson, Paris, 1994.

SIAM J. COMPUT.
Vol. 27, No. 4, pp. 942–959, August 1998
© 1998 Society for Industrial and Applied Mathematics
002

APPROXIMATION ALGORITHMS FOR THE FEEDBACK VERTEX SET PROBLEM WITH APPLICATIONS TO CONSTRAINT SATISFACTION AND BAYESIAN INFERENCE*

REUVEN BAR-YEHUDA†‡, DAN GEIGER†, JOSEPH (SEFFI) NAOR†§, AND RON M. ROTH†

Abstract. A *feedback vertex set* of an undirected graph is a subset of vertices that intersects with the vertex set of each cycle in the graph. Given an undirected graph G with n vertices and weights on its vertices, polynomial-time algorithms are provided for approximating the problem of finding a feedback vertex set of G with smallest weight. When the weights of all vertices in G are equal, the performance ratio attained by these algorithms is $4-(2/n)$. This improves a previous algorithm which achieved an approximation factor of $O(\sqrt{\log n})$ for this case. For general vertex weights, the performance ratio becomes $\min\{2\Delta^2, 4\log_2 n\}$ where Δ denotes the maximum degree in G. For the special case of planar graphs this ratio is reduced to 10. An interesting special case of weighted graphs where a performance ratio of $4-(2/n)$ is achieved is the one where a prescribed subset of the vertices, so-called *blackout* vertices, is not allowed to participate in any feedback vertex set.

It is shown how these algorithms can improve the search performance for constraint satisfaction problems. An application in the area of Bayesian inference of graphs with blackout vertices is also presented.

Key words. approximation algorithms, vertex feedback set, combinatorial optimization, Bayesian networks, constraint satisfaction

AMS subject classifications. 68Q25, 68R10, 05C85, 68T01

PII. S0097539796305109

1. Introduction.

Let $G=(V,E)$ be an undirected graph and let $w: V(G) \to \mathbb{R}^+$ be a weight function on the vertices of G. A *cycle* in G is a path whose two terminal vertices coincide. A *feedback vertex set* of G is a subset of vertices $F \subseteq V(G)$ such that each cycle in G passes through at least one vertex in F. In other words, a feedback vertex set F is a set of vertices of G such that by removing F from G, along with all the edges incident with F, a forest is obtained. A *minimum feedback vertex set of a weighted graph* (G,w) is a feedback vertex set of G of minimum weight. The weight of a minimum feedback vertex set will be denoted by $\mu(G,w)$.

The *weighted feedback vertex set (WFVS) problem* is defined as finding a minimum feedback vertex set of a given weighted graph (G,w). The special case where w is the constant function 1 is called the *unweighted feedback vertex set (UFVS) problem.* Given a graph G and an integer k, the problem of deciding whether $\mu(G,1) \le k$ is known to be NP-complete [GJ79, pp. 191–192]. Hence, it is natural to look for efficient approximation algorithms for the feedback vertex set problem, particularly

*Received by the editors February 1, 1996; accepted for publication (in revised form) April 12, 1996; published electronically May 18, 1998. A preliminary version of this paper appeared in the *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms,* Arlington, VA, 1994, pp. 344–354.

http://www.siam.org/journals/sicomp/27-4/30510.html

†Computer Science Department, Technion, Haifa 32000, Israel (dang@cs.technion.ac.il, naor@cs.technion.ac.il).

‡Part of this research was done while the author was visiting SUNY at Buffalo. This research was supported by the fund for the promotion of research at the Technion.

§The research of this author was supported in part by grant 92-00225 from the United States-Israel Binational Science Foundation (BSF), Jerusalem, Israel. Part of this research was done while the author was visiting DIMACS, Rutgers University, NJ.

in view of the recent applications of such algorithms in artificial intelligence, as we show in the sequel.

Suppose A is an algorithm that finds a feedback vertex set F_A for any given undirected weighted graph (G, w). We denote the sum of weights of the vertices in F_A by $w(F_A)$. The *performance ratio of A for (G, w)* is defined by $R_A(G, w) = w(F_A)/\mu(G, w)$. When $\mu(G, w) = 0$ we define $R_A(G, w) = 1$ if $w(F_A) = 0$ and $R_A(G, w) = \infty$ if $w(F_A) > 0$. The *performance ratio $r_A(n, w)$ of A for w* is the supremum of $R_A(G, w)$ over all graphs G with n vertices and for the same weight function w. When w is the constant function 1, we call $r_A(n, 1)$ the *unweighted performance ratio of A*. Finally, the *performance ratio $r_A(n)$ of A* is the supremum of $r_A(n, w)$ over all weight functions w defined over graphs with n vertices.

An approximation algorithm for the UFVS problem that achieves an unweighted performance ratio of $2 \log_2 n$ is essentially contained in a lemma due to Erdős and Pósa [EP62]. This result was improved by Monien and Schulz [MS81], where they achieved a performance ratio of $O(\sqrt{\log n})$.

In section 2, we provide an approximation algorithm for the UFVS problem that achieves an unweighted performance ratio of at most $4 - (2/n)$. Our algorithm draws upon a theorem by Simonovits [Si67], and our analysis uses a result by Voss [Vo68]. Actually, we consider a generalization of the UFVS problem, where a prescribed subset of the vertices, called *blackout vertices*, is not allowed to participate in any feedback vertex set. This problem is a subcase of the WFVS problem wherein each allowed vertex has unit weight and each blackout vertex has infinite weight. Our interest in graphs with blackout vertices is motivated by the *loop cutset* problem and its application to the updating problem in Bayesian inference which is explored in section 4.

In section 3, we present two algorithms for the WFVS problem. We first devise a primal-dual algorithm which is based on formulating the WFVS problem as an instance of the *set cover* problem. The algorithm has a performance ratio of 10 for weighted planar graphs and $4 \log_2 n$ for general weighted graphs. This ratio is achieved by extending the Erdős-Pósa lemma to weighted graphs. The second algorithm presented in section 3 achieves a performance ratio of $2\Delta^2(G)$ for general weighted graphs, where $\Delta(G)$ is the maximum degree of G. This result is interesting for low-degree graphs.

A notable application of approximation algorithms for the UFVS problem in artificial intelligence due to Dechter and Pearl is as follows [DP87, De90]. We are given a set of variables $x_1, x_2, \ldots, x_n$, where each x_i takes its values from a finite domain D_i. Also, for every $i < j$ we are given a constraint subset $R_{i,j} \subseteq D_i \times D_j$ which defines the allowable pairs of values that can be taken by the pair of variables (x_i, x_j). Our task is to find an assignment for all variables such that all the constraints $R_{i,j}$ are satisfied. With each instance of the problem we can associate an undirected graph G whose vertex set is the set of variables, and for each constraint $R_{i,j}$ which is *strictly* contained in $D_i \times D_j$ (i.e., $R_{i,j} \neq D_i \times D_j$) there is an edge in G connecting x_i and x_j. The resulting graph G is called a *constraint network* and it is said to represent a *constraint satisfaction problem*.

A common method for solving a constraint satisfaction problem is by backtracking, that is, by repeatedly assigning values to the variables in a predetermined order and then backtracking whenever reaching a dead end. This approach can be improved as follows. First, find a feedback vertex set of the constraint network. Then, arrange the variables so that variables in the feedback vertex set precede all other variables and

apply the backtracking procedure. Once the values of the variables in the feedback vertex set are determined by the backtracking procedure, the algorithm switches to a polynomial-time procedure SOLVE-TREE that solves the constraint satisfaction problem in the remaining forest. If SOLVE-TREE succeeds, a solution is found; otherwise, another backtracking phase occurs.

The complexity of the above modified backtracking algorithm grows exponentially with the size of the feedback vertex set: if a feedback vertex set contains k variables, each having a domain of size 2, then the procedure SOLVE-TREE might be invoked up to 2^k times. A procedure SOLVE-TREE that runs in polynomial time was developed by Dechter and Pearl, who also proved the optimality of their tree algorithm [DP88]. Consequently, our approximation algorithm for finding a small feedback vertex set reduces the complexity of solving constraint satisfaction problems through the modified backtracking algorithm. Furthermore, if the domain size of the variables varies, then SOLVE-TREE is called a number of times which is bounded from above by the product of the domain sizes of the variables whose corresponding vertices participate in the feedback vertex set. If we take the logarithm of the domain size as the weight of a vertex, then solving the WFVS problem with these weights optimizes the complexity of the modified backtracking algorithm in the case where the domain size is allowed to vary.

2. The UFVS problem. The best approximation algorithm prior to this work for the UFVS problem attained a performance ratio of $O(\sqrt{\log n})$ [MS81]. We now use some results of [Si67] and [Vo68] in order to obtain an approximation algorithm for the UFVS problem which attains a performance ratio ≤ 4. In fact, we actually consider a slight generalization of the UFVS problem where we mark each vertex of a graph as either an *allowed vertex* or a *blackout vertex*. In such graphs, feedback vertex sets cannot contain any blackout vertices. We denote the set of allowed vertices in G by $A(G)$ and the set of blackout vertices by $B(G)$. Note that when $B(G) = \emptyset$, this problem reduces to the UFVS problem. A feedback vertex set can be found in a graph G with blackout vertices if and only if every cycle in G contains at least one allowed vertex. A graph G with this property will be called a *valid graph*. The motivation for dealing with this modified problem is clarified in section 4 where we use the algorithm developed herein to reduce the computational complexity of Bayesian inference.

Throughout this section, G denotes a valid graph with a nonempty set of vertices $V(G)$ which is partitioned into a nonempty set $A(G)$ of allowed vertices, a possibly empty set $B(G)$ of blackout vertices, and a set of edges $E(G)$ possibly with parallel edges and self-loops. We use $\mu_a(G)$ as a shorthand notation for $\mu(G, w)$ where w assigns unit weight to each allowed vertex and an infinite weight to each blackout vertex. A *neighbor* of v is a vertex $u \in V(G)$ which is connected to v by an edge in $E(G)$. The degree $\Delta_G(v)$ of v in G is the number of edges that are incident with v in G. A self-loop at a vertex v contributes 2 to the degree of v. The degree of G, denoted $\Delta(G)$, is the largest among all degrees of vertices in G. A vertex in G of degree 1 is called an *endpoint*. A vertex of degree 2 is called a *linkpoint* and a vertex of any higher degree is called a *branchpoint*. A graph G is called *rich* if every vertex in $V(G)$ is a branchpoint. The notation $\Delta_a(G)$ will stand for the largest among all degrees of vertices in $A(G)$ (a degree of a vertex in $A(G)$ takes into account all incident edges, including those that lead to neighbors in $B(G)$). In a rich valid graph we have $\Delta_a(G) \geq 3$.

Two cycles in a valid graph G are *independent* if their vertex sets share only blackout vertices. Note that the size of any feedback vertex set of G is bounded from

below by the largest number of pairwise independent cycles that can be found in G. A cycle Γ in G is called *simple* if it visits every vertex in $V(G)$ at most once. Clearly, a set F is a feedback vertex set of G if and only if it intersects with every simple cycle in G. A graph is called a *singleton* if it contains only one vertex. A singleton is called *self-looped* if it contains at least one self-loop; for a singleton we have $\mu(G,1) = 1$ if it is self-looped and $\mu(G,1) = 0$ otherwise.

A graph G is connected if for every two vertices there is a connecting path in G. Every graph G can be uniquely decomposed into isolated connected components $G_1, G_2, \ldots, G_k$. Similarly, every feedback vertex set F of G can be partitioned into feedback vertex sets $F_1, F_2, \ldots, F_k$ such that F_i is a feedback vertex set of G_i. Hence, $\mu_a(G) = \sum_{i=1}^{k} \mu_a(G_i)$.

A 2-3-*subgraph* of a valid graph G is a subgraph H of G such that the degree in H of every vertex in $A(G)$ is either 2 or 3. The degree of a vertex belonging to $B(G)$ in H is not restricted. A 2-3-subgraph exists in any valid graph which is not a forest. A *maximal* 2-3-*subgraph* of G is a 2-3-subgraph of G which is not a subgraph of any other 2-3-subgraph of G. A maximal 2-3-subgraph can be easily found by applying depth-first search (DFS) on G.

A linkpoint v in a 2-3-subgraph H is called a *critical linkpoint* if v is an allowed vertex, and there is a cycle Γ in G such that $V(\Gamma) \cap V(H) \cap A(G) = \{v\}$. We refer to such a cycle Γ in G as a *witness cycle of* v. Note that we can assume a witness cycle to be simple and, so, verifying whether a linkpoint v in H is a critical linkpoint is easy: remove the set of vertices $(V(H) \cap A(G)) - \{v\}$ from G, with all incident edges, and apply a breadth-first search (BFS) to check whether there is a cycle through v in the remaining graph.

A cycle in a valid graph G is *branchpoint-free* if it does not pass through any allowed branchpoints; that is, a branchpoint-free cycle passes only through allowed linkpoints and blackout vertices of G.

The rest of this section is devoted to showing that the following algorithm correctly outputs a vertex feedback set and achieves an unweighted performance ratio less than 4.

ALGORITHM SUBG-2-3(*Input:* valid graph G; *Output:* feedback vertex set F of G);
if G is a forest then
    $F \leftarrow \emptyset$;
else begin:
    Using DFS, find a maximal 2-3-subgraph H of G;
    Using BFS, find the set X of critical linkpoints in H;
    Let Y be the set of allowed branchpoints in H;
    Find a set W that covers all branchpoint-free cycles of H which
        are not covered by X;
    $F \leftarrow X \cup Y \cup W$;
end.

Note that if $B(G) = \emptyset$, then all branchpoint-free cycles are isolated cycles in H and so W consists of one vertex of each such cycle.

We elaborate on how the set W is computed when $B(G) \neq \emptyset$. Let H' be a graph obtained from H by removing the set X along with its incident edges. Let H_b be the subgraph of H' induced by the allowed linkpoints and blackout vertices of H'. For every isolated cycle in H_b, we arbitrarily choose an allowed linkpoint from that cycle to W. Next, we replace each maximal (with respect to containment) chain of allowed

linkpoints in H_b by an edge, resulting in a graph H_b^*. We assign unit cost to all edges corresponding to a chain of allowed linkpoints, and a zero cost to all other edges, and compute a minimum-cost spanning forest T of H_b^*. We now add to W one linkpoint from each chain of allowed linkpoints in H_b that corresponds to an edge in $H_b^* - T$. It is now straightforward to verify that the complexity of SubG-2-3 is linear in $|E(G)|$.

The following two lemmas, which generalize some claims used in the proof of Theorem 1 in [Si67], are used to prove that SubG-2-3 outputs a feedback vertex set of a valid graph G.

LEMMA 1. *Let H be a maximal 2-3-subgraph of a valid graph G and let Γ be a simple cycle in G. Then, one of the following holds:*

(a) *Γ is a witness cycle of some critical linkpoint of H or*

(b) *Γ passes through some allowed branchpoint of H or*

(c) *Γ is a cycle in H that consists only of blackout vertices or allowed linkpoints of H.*

Proof. Let Γ be a simple cycle in G and assume to the contrary that neither of (a)–(c) holds. This implies in particular that Γ cannot be entirely contained in H. We distinguish between two cases: (1) Γ does not intersect with H and (2) Γ intersects with H only in blackout vertices and allowed linkpoints of H.

Case 1. In this case we could join Γ and H to obtain a 2-3-subgraph H^* of G that contains H as a proper subgraph. This however contradicts the maximality of H.

Case 2. If Γ intersects with H only in blackout vertices, then as in Case 1, we can join Γ and H and contradict the maximality of H. Suppose now that Γ intersects with H in some allowed linkpoints of H. Note that in such a case Γ must intersect with H in at least two distinct allowed linkpoints of H, or else Γ would be a witness cycle of the only intersecting (critical) linkpoint. Since Γ is not contained in H by assumption, we can find two allowed linkpoints v_1 and v_2 in $V(\Gamma) \cap V(H)$ that are connected by a path P along Γ such that $V(P) \cap V(H) \cap A(G) = \{v_1, v_2\}$ and P is not entirely contained in H. Joining P and H, we obtain a 2-3-subgraph of G that contains H as a proper subgraph, thus contradicting the maximality of H. □

LEMMA 2. *Let H be a maximal 2-3-subgraph of G and let Γ_1 and Γ_2 be witness cycles in G of two distinct critical linkpoints in H. Then Γ_1 and Γ_2 are independent cycles, namely, $V(\Gamma_1) \cap V(\Gamma_2) \subseteq B(G)$.*

Proof. Let v_1 and v_2 be the critical linkpoints associated with Γ_1 and Γ_2, respectively, and assume to the contrary that $V(\Gamma_1) \cap V(\Gamma_2)$ contains an allowed vertex $u \in A(G)$. Then, there is a path P in G that runs along parts of the cycles Γ_1 and Γ_2, starting from v_1, passing through u, and ending at v_2. Since Γ_1 and Γ_2 are witness cycles, we have $V(P) \cap V(H) \cap A(G) = \{v_1, v_2\}$. And, since v_1 and v_2 are distinct critical linkpoints, the vertex u cannot possibly coincide with either of them. Therefore, the path P is not entirely contained in H. Joining P and H we obtain a 2-3-subgraph of G that contains H as a proper subgraph, thus reaching a contradiction. □

THEOREM 3. *For every valid graph G, the set F computed by* SubG-2-3 *is a feedback vertex set of G.*

Proof. Let Γ be a simple cycle in G. We follow the three cases of Lemma 1 to show that $V(\Gamma) \cap F \neq \emptyset$.

(a) *Γ is a witness cycle of some critical linkpoint of H.* By construction, all critical linkpoints of H are in F.

(b) *Γ passes through some allowed branchpoint of H.* By construction, all allowed branchpoints of H are in F.

(c) Γ *is a cycle in* H *that consists only of blackout vertices or allowed linkpoints of* H. When $V(\Gamma)$ contains a critical linkpoint, then SubG-2-3 selects that linkpoint into the feedback vertex set F. Otherwise, the cycle Γ must be entirely contained in the graph H_b that was used to create W. We now show that W covers all cycles in H_b. Assume the contrary and let Γ be a cycle in H_b that is not covered by W. Recall that in the construction of H_b^*, each chain of allowed linkpoints in Γ was replaced by an edge with a unit cost. Let Γ^* be the resulting cycle in H_b^*. Since W does not cover Γ, all unit-cost edges in Γ^* were necessarily chosen to the minimum-cost spanning forest T. On the other hand, since T does not contain any cycles, there must be at least one zero-cost edge of Γ^* which is not contained in T. Hence, by deleting one of the unit-cost edges of Γ^* from T and inserting instead a particular zero-cost edge of Γ^* into T, we can obtain a new spanning forest T' for H_b^*. However, the cost of T' is smaller than that of T, which contradicts our assumption that T is a minimum-cost spanning forest. □

A *reduction graph* G' of an undirected graph G is a graph obtained from G by a sequence of the following transformations:

- Delete an endpoint and its incident edge.
- Connect two neighbors of a linkpoint v (other than a self-looped singleton) by a new edge and remove v from the graph with its two incident edges.

A reduction graph of a valid graph G is not necessarily valid, since the reduction process may generate a cycle consisting of blackout vertices only. We will be interested in reduction sequences in which each transformation yields a valid graph.

Lemma 4. *Let* G' *be a reduction graph of* G. *If* G' *is valid, then* $\mu_a(G') = \mu_a(G)$.

Proof. Let $H_1 = G, H_2, \ldots, H_{t-1}, H_t = G'$ be a sequence of reduction graphs where each H_i is obtained by a removal of one linkpoint and possibly some endpoints from H_{i-1}. Since G' is valid, each H_i is a valid graph as well. Let v_i be the linkpoint that is removed from H_i to obtain H_{i+1}.

First we show that $\mu_a(G') \geq \mu_a(G)$. Suppose F is a feedback vertex set of H_{i+1} for some i, $1 \leq i < t$ and let Γ be a cycle in H_i that passes through v_i. A reduction of Γ obtained by replacing the linkpoint v_i on Γ by an edge connecting the neighbors of v_i yields a cycle $\hat{\Gamma}$ in H_{i+1}. The vertex set of $\hat{\Gamma}$ intersects the set F. Hence, F is also a feedback vertex set of H_i which implies that $\mu_a(H_{i+1}) \geq \mu_a(H_i)$.

Now we show that $\mu_a(G') \leq \mu_a(G)$. Suppose F is a minimal feedback vertex set of H_i. If F does not contain v_i, then F is also a feedback vertex set of H_{i+1}. Otherwise, write $F = \{v_i\} \cup F'$. We claim that the set F' cannot fail to cover more than one cycle in H_{i+1}. If it failed, then there would be two distinct cycles Γ_1 and Γ_2 in H_i that contain v_i, in which case the cycle in H_i induced by $(V(\Gamma_1) \cup V(\Gamma_2)) - \{v_i\}$ would not be covered by F, thus contradicting the fact that F is a feedback vertex set of H_i. It follows by this and the minimality of F that the set F' fails to cover *exactly* one cycle in H_{i+1}. This cycle contains at least one allowed vertex u because H_{i+1} is a valid graph. Therefore, the set $F' \cup \{u\}$ is a feedback vertex set of H_{i+1}. Hence, $\mu_a(H_{i+1}) \leq \mu_a(H_i)$. □

A reduction graph G^* of a graph G is *minimal* if and only if G^* is a valid graph and any proper reduction graph G' of G^* is not valid.

Lemma 5. *If* G^* *is a minimal reduction graph of* G, *then* G^* *does not contain blackout linkpoints, and every feedback vertex set of* G^* *contains all allowed linkpoints of* G^*.

Proof. Recall that G^* is a valid graph. If G^* contains a blackout linkpoint, then its removal creates a valid reduction graph which contradicts the minimality of G^*.

Now assume F is a feedback vertex set of G^* and v is an allowed linkpoint which is not in F. If the removal of v yields a graph that is not valid, then v must have been included in F. If the removal of v yields a valid graph, then G^* is not minimal. □

The next lemma is needed in order to establish the performance ratio of SubG-2-3. It is a variant of Lemma 4 in [Vo68].

LEMMA 6. *Let G be a valid graph with no blackout linkpoints and such that no vertex has degree less than 2. Then, for every feedback vertex set F of G which contains all linkpoints of G,*

$$|V(G)| \;\leq\; (\Delta_a(G)+1)\,|F| \;-\; 2\,.$$

Proof. Suppose $F = V(G)$. In this case we have $|V(G)| \leq 3|V(G)| - 2 \leq (\Delta_a(G)+1)|V(G)| - 2$ and, therefore, the lemma holds trivially. So we assume from now on that $|F| < |V(G)|$.

Let E_F denote the set of edges in $E(G)$ whose terminal vertices are all vertices in F. Define $X = V - F$ and let E_X denote the set of edges in $E(G)$ whose terminal vertices are all vertices in X. Also, let $E_{F,X}$ denote the set of those edges in G that connect vertices in F with vertices in X. Clearly, E_F, E_X, and $E_{F,X}$ form a partition on $E(G)$. Now, the graph obtained by deleting F from G is a nonempty forest on X and, therefore, $|E_X| \leq |X| - 1$. However, each vertex in X is a branchpoint in G because all linkpoints are assumed to be in F and there are no vertices of degree less than 2. Therefore,

$$3\,|X| \;\leq\; \sum_{v\in X} \Delta_G(v) \;=\; |E_{F,X}| \;+\; 2\,|E_X| \;\leq\; |E_{F,X}| \;+\; 2\,(|X|-1)$$

i.e.,

$$|E_{F,X}| \;\geq\; |X| \;+\; 2 \;=\; |V(G)| \;-\; |F| \;+\; 2\,.$$

On the other hand,

$$\Delta_a(G)\,|F| \;\geq\; \sum_{v\in F} \Delta_G(v) \;=\; |E_{F,X}| \;+\; 2\,|E_F|\,.$$

Combining the last two inequalities we obtain

$$|V(G)| \;\leq\; (\Delta_a(G)+1)\,|F| \;-\; 2\,|E_F| \;-\; 2\,. \qquad \square$$

The main claim of this section now follows.

THEOREM 7. *The unweighted performance ratio of* SubG-2-3 *is at most* $4 - (2/|V(G)|)$.

Proof. Let F be the feedback vertex set computed by SubG-2-3 for a valid graph G which is not a forest. We show that $|F| \;\leq\; 4\,\mu_a(G) \;-\; 2$. The theorem follows immediately from this inequality.

Let H, X, Y, and W be as in SubG-2-3. Suppose $\mu_a(G) = 1$. Then, all cycles in G pass through some allowed vertex v in G and, so, no vertex other than v can be a critical linkpoint in H. Now, if v is a linkpoint in H, then H is a cycle. Otherwise, one can readily verify that H must contain exactly two branchpoints. In either case we have $|F| \leq 2$. We assume from now on that $\mu_a(G) \geq 2$.

For every $v_i \in X$, let Γ_i be some witness cycle of v_i in G. By Lemma 2, the cycles Γ_i are pairwise independent. Therefore, the minimum number of vertices needed to cover such cycles is $|X|$.

Let $\{\Gamma_j^*\}$ be the set of branchpoint-free cycles in H that do not contain any critical linkpoints of H. Note that each cycle Γ_j^* is independent with any witness cycle Γ_i. We now claim that any smallest set W' of vertices of $V(H)$ that intersects with the vertex set of each Γ_j^* must be of size $|W|$. To see this, note that W' contains only allowed linkpoints of H. If we remove from H_b^* all the edges that correspond to linkpoints belonging to W', then we clearly end up with a forest. By construction, the minimum number of edges (or allowed linkpoints) needed to be removed from H_b^* so as to make it into a forest is $|W|$.

Recalling that every cycle Γ_j^* is independent with any witness cycle Γ_i, the set W' cannot possibly intersect with any of the cycles Γ_i. Hence, in order to cover the cycles $\{\Gamma_i\} \cup \{\Gamma_j^*\}$ in G, we will need at least $|X| + |W'|$ vertices. Therefore,

$$\mu_a(G) \;\geq\; |X| + |W'| \;=\; |X| + |W| \;.$$

On the other hand, we recall that $|F| = |X| + |Y| + |W|$.

We distinguish between the following two cases.

Case 1. $|Y| \leq 2|X| + 2|W|$. Here we have

$$|F| \;=\; |X| + |Y| + |W| \;\leq\; 3|X| + 3|W| \;\leq\; 3\mu_a(G) \;\leq\; 4\mu_a(G) - 2 \;.$$

Case 2. $|Y| > 2|X| + 2|W|$. Let F^* be a feedback vertex set of G of size $\mu_a(G)$ and let W' be a smallest subset of F^* that intersects with the vertex set of each Γ_j^*. Clearly, W' consists of allowed linkpoints of H, and, as we showed earlier in this proof, $|W'| = |W|$. Let H_1 be the subgraph of H obtained by removing all critical linkpoints of H and all linkpoints in W'. With each deleted linkpoint, we also remove recursively all resulting endpoints from H while obtaining H_1. Thus, a deletion of a linkpoint from H can decrease the number of branchpoints by 2 at most. Hence, the number of branchpoints left in H_1 is at least $|Y| - 2|X| - 2|W| > 0$. Furthermore, the graph H_1 does not contain any endpoints.

Let H_1^* be a minimal reduction graph of H_1 and let H_2 be a valid graph obtained by removing all singleton components from H_1^*. Since H_1 does not contain any endpoints, the number of branchpoints of H_1 is preserved in H_1^* and in H_2. Therefore, the graph H_2 contains at least $|Y| - 2|X| - 2|W|$ branchpoints. On the other hand, since H_1^* is a minimal reduction and due to Lemma 5, there are no blackout linkpoints in H_1^* and every feedback vertex set of H_1^* contains all allowed linkpoints of H_1^*. Furthermore, the graphs H_1^* and H_2 do not contain any endpoints.

It follows that we can apply Lemma 6 to H_2 and any feedback vertex set of H_2, thus obtaining

$$|Y| - 2|X| - 2|W| \;\leq\; 4\mu_a(H_2) - 2 \;\leq\; 4\mu_a(H_1^*) - 2 \;=\; 4\mu_a(H_1) - 2 \;,$$

where the equality is due to Lemma 4. Therefore,

$$\begin{aligned} |F| &= |X| + |Y| + |W| \\ &\leq 4|X| + 4|W| + |Y| - 2|X| - 2|W| \\ &\leq 4(|X| + |W| + \mu_a(H_1)) - 2 \;. \end{aligned} \tag{1}$$

Recall that W' was chosen as a subset of a smallest feedback vertex set F^* of G. Let X' be a smallest subset of F^* that covers the witness cycles $\{\Gamma_i\}$ and let Z' be a smallest subset of F^* that covers the cycles of H_1. Since H_1 does not contain any of the critical linkpoints of H, each witness cycle Γ_i is independent with any cycle in

H_1 and, so, we have $X' \cap Z' = \emptyset$. It also follows from our previous discussion that $X' \cap W' = \emptyset$. In addition, by construction of H_1 we have $W' \cap Z' = \emptyset$. It thus follows that

$$|X| + |W| + \mu_a(H_1) \;\leq\; |X'| + |W'| + |Z'| \;\leq\; |F^*| \;=\; \mu_a(G)\ .$$

Combining with (1), we obtain the desired result. □

3. WFVS. In this section, we consider the approximation of the WFVS problem described in section 1. That is, given an undirected graph G and a weight function w on its vertices, find a feedback vertex set of (G, w) with minimum weight. As in the previous section, we assume that G may contain parallel edges and self-loops.

A *weighted reduction graph* G' of an undirected graph G is a graph obtained from G by a sequence of the following transformations.

- Delete an endpoint and its incident edge.
- Let u and v be two adjacent vertices such that $w(u) \leq w(v)$ and v is a linkpoint. Connect u to the other neighbor of v, and remove v from the graph with its two incident edges.

The following lemma can be easily verified. (See, e.g., the proof of Lemma 4.)

LEMMA 8. *Let (G', w') be a weighted reduction graph of (G, w). Then, $\mu(G', w') = \mu(G, w)$.*

A weighted reduction graph G^* of a graph G is *minimal* if and only if any weighted reduction graph G' of G^* is equal to G^*. A graph is called *branchy* if it has no endpoints and, in addition, its set of linkpoints induces an independent set; i.e., each linkpoint is either an isolated self-looped singleton or connected to two branchpoints. Clearly, any minimal weighted reduction graph must be branchy. We note that the complexity of transforming a graph into a branchy graph is linear in $|E(G)|$.

We are now ready to present our algorithms for finding an approximation for a minimum-weight feedback vertex set of a given weighted graph. In section 3.1 we give an algorithm that achieves a performance ratio of $4 \log_2 |V(G)|$. In section 3.2 we present an algorithm that achieves a performance ratio of $2\Delta^2(G)$.

3.1. The primal-dual algorithm. The basis of the first approximation algorithm is the next lemma which generalizes a lemma due to Erdős and Pósa [EP62, Lemma 3]. That lemma was obtained by Erdős and Pósa while estimating the smallest number of edges in a graph which contains a given number of pairwise independent cycles. Later on, in [EP64], they provided bounds on the value of $\mu(G, 1)$ in terms of the largest number of pairwise independent cycles in G. Tighter bounds on $\mu(G, 1)$ were obtained by Simonovits [Si67] and Voss [Vo68].

LEMMA 9. *The shortest cycle in any branchy graph G with at least two vertices is of length $\leq 4 \log_2 |V(G)|$.*

Proof. Let t be the smallest even integer such that $2 \cdot 2^{t/2} > |V(G)|$. Apply BFS on G of depth t starting at some vertex v. We now claim that the search hits some vertex twice and so there exists a cycle of length $\leq 2t$ in G. Indeed, if it were not so, then the induced BFS tree would contain at least $2 \cdot 2^{t/2}$ distinct vertices of G, which is a contradiction. □

In each iteration of the proposed algorithm, we first find a minimal weighted reduction graph, and then find a cycle Γ with the smallest number of vertices in the minimal weighted reduction graph. The algorithm sets δ to be the minimum among the weights of the vertices in $V(\Gamma)$. This value of δ is subtracted, in turn, from the weight of each vertex in $V(\Gamma)$. Vertices whose weight becomes zero are added to the

feedback vertex set and deleted from the graph. Each such iteration is repeated until the graph is exhausted.

ALGORITHM MINIWCYCLE(*Input:* (G, w); *Output:* feedback vertex set F of (G, w));
$F \leftarrow \emptyset$; $(H, w_H) \leftarrow (G, w)$;
While H is not a forest do begin:
  Find a minimal weighted reduction graph $(H', w_{H'})$ of (H, w_H);
  Find a cycle Γ' in H' with the smallest number of vertices;
  Set $\delta \leftarrow \min_{v \in V(\Gamma')} w_{H'}(v)$;
  Set $w_{H'}(v) \leftarrow w_{H'}(v) - \delta$ for every $v \in V(\Gamma')$;
  Let $X = \{v \in V(\Gamma') \ : \ w_{H'}(v) = 0\}$;
  Remove X (with all incident edges) from H';
  $(H, w_H) \leftarrow (H', w_{H'})$;
  $F \leftarrow X \cup F$;
end.

Finding a shortest cycle can be done by running BFS from each vertex until a cycle is found and then selecting the smallest. A more efficient approach for finding the shortest cycle is described in [IR78].

It is not hard to see that MINIWCYCLE computes a feedback vertex set of G. We now analyze the algorithm employing techniques similar to those used in [Ho82], [Ho83], and [KhVY94]. We note that the algorithm can also be analyzed using the local ratio theorem of Bar-Yehuda and Even [BaEv85].

THEOREM 10. *The performance ratio of algorithm* MINIWCYCLE *is at most* $4 \log_2 |V(G)|$.

Proof. We assume that $|V(G)| > 1$. Given a feedback vertex set F of (G, w), let $\mathbf{x} = [x_v]_{v \in V(G)}$ be the indicator vector of F, namely, $x_v = 1$ if $v \in F$ and $x_v = 0$ otherwise. We denote by $\mathcal{C}$ the set of cycles in G. The problem of finding a minimum-weight feedback vertex set of (G, w) can be formulated in terms of $\mathbf{x}$ by an integer programming problem as follows:

$$\begin{gathered} \text{minimize} \quad \textstyle\sum_{v \in V(G)} w(v) \cdot x_v \\ \text{ranging over all nonnegative integer vectors } \mathbf{x} = [x_v]_{v \in V(G)} \text{ such that} \\ \displaystyle\sum_{v \in V(\Gamma)} x_v \ \geq \ 1 \quad \text{for every} \quad \Gamma \in \mathcal{C} \ . \end{gathered} \tag{2}$$

Let $\mathcal{C}_v$ denote the set of cycles passing through vertex v in G and consider the following integer programming *packing* problem:

$$\begin{gathered} \text{maximize} \quad \textstyle\sum_{\Gamma \in \mathcal{C}} y_\Gamma \\ \text{ranging over all nonnegative integer vectors } \mathbf{y} = [y_\Gamma]_{\Gamma \in \mathcal{C}} \text{ such that} \\ \displaystyle\sum_{\Gamma \in \mathcal{C}_v} y_\Gamma \ \leq \ w(v) \quad \text{for every} \quad v \in V \ . \end{gathered} \tag{3}$$

Clearly, the linear relaxation of (3) is the dual of the linear relaxation of (2), with y_Γ, $\Gamma \in \mathcal{C}$, being the dual variables.

Let $(H', w_{H'})$ be a minimal weighted reduction graph computed at some iteration of algorithm MINIWCYCLE. Then, for each cycle $\Gamma' \in H'$, we associate a unique cycle $\Gamma \in G$ as follows: if all vertices in $V(\Gamma')$ belong to G, then $\Gamma = \Gamma'$. Otherwise, we "unfold" the transformation steps performed in obtaining H' from H in backward order, i.e., from H' back to H: in each such step we add to Γ' chains of linkpoints

(connecting vertices in Γ') that were deleted. When this process finishes, the cycle Γ' of H' transforms into a cycle Γ of G.

We now show that MiniWCycle can be interpreted as a primal-dual algorithm. We first show that it computes a dual feasible solution for (3) with a certain maximality property. The initial dual feasible solution is the one in which all the dual variables y_Γ are zero.

Let Γ'_i be a cycle chosen at iteration i of MiniWCycle and let Γ_i be the associated cycle in G. We may view the computation of iteration i of MiniWCycle as setting the value of the dual variable y_{Γ_i} to the weight δ of a lightest vertex in $V(\Gamma'_i)$. The updated weight $w_{H'}(v)$ of every $v \in V(\Gamma'_i)$ is precisely the slack of the dual constraint

$$\sum_{\Gamma \in \mathcal{C}_v} y_\Gamma \;\leq\; w(v) \tag{4}$$

that corresponds to v.

It is clear that by the choice of δ, the values of the dual variables y_Γ at the end of iteration i of MiniWCycle satisfy the dual constraints (4) corresponding to vertices $v \in V(\Gamma'_i)$. It thus follows that the dual constraints hold for all vertices $v \in V(H')$ at iteration i.

Let v be a vertex that was removed from H to obtain H' in iteration i of MiniWCycle. It remains to show that the dual constraint (4) corresponding to such a vertex holds in each iteration j of the algorithm for every $j \geq i$.

We show this by backward induction on j. By the previous discussion it follows that the constraints corresponding to vertices that exist in the last iteration all hold. Suppose now that the dual constraints corresponding to vertices in $V(H')$ in iteration j are not violated. We show that the dual constraints corresponding to vertices in $V(H) - V(H')$ in that iteration are also not violated. Let c be a chain of linkpoints in H in iteration j, and let v_1 and v_2 be the two branchpoints adjacent to c. Let u be a vertex of minimum weight among v_1, v_2, and the vertices in c. We note that the weighted reduction procedure deletes all vertices in c except possibly for one representative, depending on whether u is in c or is one of its adjacent branchpoints. We now observe that the set of cycles that passes through a linkpoint in c is the same for all linkpoints in c and is contained in the set of cycles that pass through v_1 and is also contained in the set of cycles that pass through v_2. This implies that if the dual constraint corresponding to u is not violated, then the dual constraints corresponding to any vertex in c is also not violated.

The algorithm essentially constructs a primal solution $\mathbf{x}$ from the dual solution $\mathbf{y}$: it selects into the feedback vertex set all vertices for which (i) the corresponding dual constraints are tight and (ii) in the iteration the constraint first became tight, the corresponding vertex belonged to the graph. As stated earlier, this construction yields a feasible solution.

Let $\mathbf{x}^* = [x^*_v]_{v \in V(G)}$ and $\mathbf{y}^* = [y^*_\Gamma]_{\Gamma \in \mathcal{C}}$ denote the optimal primal and dual fractional solutions, respectively. It follows from the duality theorem that

$$\sum_{v \in V(G)} w(v) \cdot x_v \;\geq\; \sum_{v \in V(G)} w(v) \cdot x^*_v \;=\; \sum_{\Gamma \in \mathcal{C}} y^*_\Gamma \;\geq\; \sum_{\Gamma \in \mathcal{C}} y_\Gamma \,. \tag{5}$$

Hence, to prove the theorem it suffices to bound the ratio between the LHS and the RHS of (5). First note that $y_\Gamma \neq 0$ only for cycles Γ in G that are associated with cycles Γ' that were chosen at some iteration of MiniWCycle. By the above

construction of $\mathbf{x}$, it is clear that the dual variable y_Γ of each such cycle Γ contributes its value to at most $V(\Gamma')$ vertices. Hence,

$$\sum_{v \in V(G)} w_v \cdot x_v \;\leq\; \sum_{v \in V(G)} \sum_{\Gamma \in \mathcal{C}_v} y_\Gamma \;\leq\; \sum_{\Gamma \in \mathcal{C}} y_\Gamma \cdot |V(\Gamma')| \,.$$

Now, in each iteration, the graph H' is a branchy graph. Therefore, by Lemma 9, we have that $|V(\Gamma')| \leq 4 \log_2 |V(G)|$. Hence the theorem is proved. □

PROPOSITION 11. *For planar graphs, the weighted performance ratio of* MINIWCYCLE *is at most* 10.

Proof. We first notice that the weighted reduction process preserves planarity and, therefore, at each iteration of algorithm MINIWCYCLE we remain with a planar graph.

We claim that every rich planar graph G must contain a face of length at most 5. Assume the contrary. By summing up the lengths of all the faces, we get that $2|E| \geq 6|Z|$, where Z denotes the set of faces of G. By Euler's formula,

$$|E| - |V| + 2 = |Z|.$$

Hence, $2|E| \leq 3|V| - 6$. However, since the degree of each vertex is at least 3, we get that $2|E| \geq 3|V|$, which is a contradiction. Furthermore, this implies that a branchy planar graph must contain a cycle of length at most 10. □

3.2. Low-degree graphs. The algorithm presented in this section is based on the following variant of Lemma 6.

LEMMA 12. *Let G be a branchy graph. Then, for every feedback vertex set F of G,*

$$|V(G)| \;\leq\; 2\Delta^2(G) \cdot |F|.$$

Proof. Let F be a feedback vertex set of G. We can assume without loss of generality that F contains only branchpoints, since this assumption can only decrease $|F|$. Let G' be the minimal (unweighted) reduction graph of G; i.e., G' contains only branchpoints or isolated self-looped singletons. Clearly, F is also a feedback vertex set of G'. Thus, G' and F satisfy the conditions of Lemma 6 ($\Delta_a = \Delta$), yielding that

$$|V(G')| \;\leq\; (\Delta(G') + 1) \cdot |F| \;-\; 2 \,.$$

Since G' is a branchy graph, the number of linkpoints in G can be at most $\Delta(G') \cdot |V(G')|/2$. Hence,

$$|V(G)| \;\leq\; \frac{(\Delta(G) + 2) \cdot |V(G')|}{2}$$

and, so,

$$|V(G)| \;\leq\; \frac{(\Delta(G) + 2) \cdot (\Delta(G) + 1) \cdot |F|}{2} \;\leq\; 2\Delta^2(G) \cdot |F|. \qquad \square$$

We now present a weighted greedy algorithm for finding a feedback vertex set in a graph G.

ALGORITHM WGREEDY(*Input:* (G, w); *Output:* feedback vertex set F of (G, w));
$F \leftarrow \emptyset$; $i \leftarrow 1$; $(H, w_H) \leftarrow (G, w)$;
while H is not a forest do begin:
    Find a minimal weighted reduction graph $(H'_i, w_{H'_i})$ of (H, w_H);
    $\alpha_i \leftarrow \min_{v \in V(H'_i)} w_{H'_i}(v)$;
    $U_i \leftarrow \{u \in V(H'_i) \mid w_{H'_i}(u) = \alpha_i\}$;
    $F \leftarrow F \cup U_i$;
    remove U_i from H'_i with its incident edges;
    $(H, w_H) \leftarrow (H'_i, w_{H'_i})$;
    $i \leftarrow i + 1$;
end.

For a subset $S \subseteq V$, let $w(S)$ denote the sum of weights of the vertices in S. We now prove the following theorem.

THEOREM 13. *Let G be a branchy graph. Denote by F the feedback vertex set computed by algorithm* WGREEDY*, and by F^* a minimum-weight feedback vertex set in G. Then, $w(F) \leq 2\Delta^2(G) \cdot w(F^*)$.*

Proof. Assume that the number of iterations the *while* loop is executed in algorithm WGREEDY is p. We define the following weight functions $w_1, \ldots, w_p$ on $V(G)$. The weight function w_i is defined, for $1 \leq i \leq p$, as follows:

$$\text{for all } v \in V(G): \quad w_i(v) = \begin{cases} \alpha_i - \alpha_{i-1} & \text{if } v \in V(H'_i), \\ 0 & \text{otherwise,} \end{cases}$$

where $\alpha_0 = 0$.

For a subset S, let $w_i(S)$ denote the sum of weights of the vertices in S, where the weight function is w_i. Clearly,

$$w(F) \;=\; \sum_{i=1}^{p} w(U_i) \;=\; \sum_{i=1}^{p} w_i(F).$$

Suppose that at one of the weighted reduction steps of algorithm WGREEDY, a chain c of equal weight linkpoints was reduced to a single vertex, say, v, which either belongs to c or is one of the two branchpoints adjacent to c. Suppose further that v was added to F. If F^* also contains a vertex from the chain c, then without loss of generality we can assume that this vertex can be replaced by v.

Let $u \in F^*$. Obviously, $u \in H'_1$. We claim that if $u \notin F$, then $u \in H'_i$ for all $i = 1, 2, \ldots, p$. Assume this is not the case. Then, with respect to the order in which vertices entered F in algorithm WGREEDY, let u be the first vertex such that $u \in F$, $u \notin F^*$, and u was removed from the graph in a weighted reduction step. This means that u was at the time of its removal a linkpoint that had an adjacent vertex u' with smaller weight. But then, by exchanging u for u' in F^*, we obtain a feedback vertex set which has smaller weight, contradicting the optimality of F^*. Hence, for a vertex $u \in F^*$, $w(u) \;\geq\; \sum_{i=1}^{p} w_i(u)$. Therefore,

$$w(F^*) \;\geq\; \sum_{i=1}^{p} w_i(F^*)\,.$$

Notice that in the graph H'_i, the weight function w_i assigns the same weight to all vertices. Hence, by Lemma 12, we have that $w_i(F) \leq 2\Delta^2(H'_i) \cdot w_i(F^*)$ for all $i = 1, 2, \ldots, p$. Since $\Delta(H'_i) \leq \Delta(G)$ for all i, the theorem follows. □

It follows from Lemma 8 that the performance ratio of algorithm WGREEDY for (G, w) is at most $2\Delta^2(G)$ for *any* graph G.

4. The loop cutset problem and its application. In section 4.1 we consider a variant of the WFVS problem for directed graphs and in section 4.2 we describe its application to Bayesian inference.

4.1. The loop cutset problem. The underlying graph of a directed graph D is the undirected graph formed by ignoring the directions of the edges in D. A *loop* in D is a subgraph of D whose underlying graph is a cycle. A vertex v is a *sink* with respect to a loop Γ if the two edges adjacent to v in Γ are directed into v. Every loop must contain at least one vertex that is not a sink with respect to that loop. Each vertex that is not a sink with respect to a loop Γ is called an *allowed vertex with respect to* Γ. A *loop cutset* of a directed graph D is a set of vertices that contains at least one allowed vertex with respect to each loop in D. Our problem is to find a minimum-weight loop cutset of a given directed graph D and a weight function w. We denote by $\mu(D, w)$ the sum of weights of the vertices in such a loop cutset. Greedy approaches to the loop cutset problem have been suggested by [SuC90] and [St90]. Both methods can be shown to have a performance ratio as bad as $\Omega(n/4)$ in certain planar graphs [St90]. An application of our approximation algorithms to the loop cutset problem in the area of Bayesian inference is described later in this section.

The approach we take is to reduce the weighted loop cutset problem to the weighted feedback vertex set problem solved in the previous section. Given a weighted directed graph (D, w), we define the *splitting* weighted undirected graph (D_s, w_s) as follows. Split each vertex v in D into two vertices v_{in} and v_{out} in D_s such that all incoming edges to v become undirected incident edges with v_{in} and all outgoing edges from v become undirected incident edges with v_{out}. In addition, we connect v_{in} and v_{out} by an undirected edge. Set $w_s(v_{\text{in}}) = \infty$ and $w_s(v_{\text{out}}) = w(v)$. For a set of vertices X in D_s, we define $\psi(X)$ as the set obtained by replacing each vertex v_{in} or v_{out} in X by the respective vertex v in D from which these vertices originated.

Our algorithm can now be easily stated.

ALGORITHM LOOPCUTSET (*Input:* (D, w); *Output:* loop cutset F of (D, w));
Construct (D_s, w_s);
Apply MINIWCYCLE on (D_s, w_s) to obtain a feedback vertex set X;
$F \leftarrow \psi(X)$.

Note that each loop in D is associated with a unique cycle in D_s, and vice versa, in a straightforward manner. Let $I(\Gamma)$ denote the loop image of a cycle Γ in D_s, and $I^{-1}(K)$ denote the cycle image of a loop K in D. It is clear that the mapping I is $1-1$ and onto.

The next lemma shows that algorithm LOOPCUTSET outputs a loop cutset of (D, w).

LEMMA 14. *Let (D, w) be a directed weighted graph and (D_s, w_s) be its splitting graph. Then* (i) *if F is a feedback vertex set of (D_s, w_s) having finite weight, then $\psi(F)$ is a loop cutset of (D, w), and $w_s(F) = w(\psi(F))$;* (ii) *if U is a loop cutset of D, then the set U_s obtained from U by replacing each vertex $v \in U$ by vertex $v_{out} \in D_s$ is a feedback vertex set of D_s, and $w(U) = w_s(U_s)$.*

Proof. We prove (i). The proof of (ii) is similar. Let Γ be a loop in D. To prove the lemma we show that an allowed vertex with respect to Γ belongs to $\psi(F)$. Let $I^{-1}(\Gamma)$ be the unique cycle image of Γ in D_s. Since F is a cycle cover of D_s having finite weight, there must be a vertex $v_{\text{out}} \in F$ in $I^{-1}(\Gamma)$. Now, it is clear that vertex $v \in \Gamma$ from which v_{out} originated is an allowed vertex with respect to Γ as needed. To complete the proof, by the finiteness of $w_s(F)$ we must have $w_s(F) = w(\psi(F))$, since $w_s(v_{\text{out}}) = w(v)$ for each vertex in F. ∎

It follows from Lemma 14 that $\mu(D, w) = \mu(D_s, w_s)$. In addition, due to Theorem 10 applied to the graph D_s, and since the number of vertices in D_s is twice the number of vertices in D, we get the following bound on the performance ratio of algorithm LoopCutset.

THEOREM 15. *The performance ratio of* LoopCutset *is at most* $4\log_2(2|V(D)|)$.

We now show that in the unweighted loop cutset problem we can achieve a performance ratio better than 4. In this case, for each vertex $v \in D$, the weight of $v_{\text{in}} \in D_s$ is one unit, and the weight of $v_{\text{out}} \in D_s$ is ∞. This falls within the framework considered in section 2, since vertices with infinite weight in D_s can be treated as blackout vertices. We can therefore apply SubG-2-3 in the LoopCutset algorithm instead of applying MiniWCycle and obtain the following improved performance ratio.

THEOREM 16. *When using* SubG-2-3*, the unweighted performance ratio of* LoopCutset *is at most* $4 - (2/|V(D)|)$.

Proof. We have

$$w(\psi(F)) = w_s(F) \leq 4\mu(D_s, w_s) - 2,$$

where the equality is due to Lemma 14, and the inequality is due to Theorem 7. Since $\mu(D_s, w_s) = \mu(D, w) \leq |V(D)|$, the claim is proved. ∎

4.2. An application. We conclude this section with an application of approximation algorithms for the loop cutset problem.

Let $P(u_1, \ldots, u_n)$ be a probability distribution where each u_i draws values from a finite set called the *domain* of u_i. A directed graph D with no directed cycles is called a *Bayesian network of* P if there is a 1-1 mapping between $\{u_1, \ldots, u_n\}$ and vertices in D such that u_i is associated with vertex i and P can be written as follows:

$$P(u_1, \ldots, u_n) = \prod_{i=1}^{n} P(u_i \mid u_{i_1}, \ldots, u_{i_{j(i)}}) \,, \tag{6}$$

where $i_1, \ldots, i_{j(i)}$ are the source vertices of the incoming edges to vertex i in D.

It is worth noting that Bayesian networks are useful knowledge representation schemes for many artificial intelligence tasks. Bayesian networks allow a wide spectrum of independence assumptions to be considered by a model builder so that a practical balance can be established between computational needs and adequacy of conclusions. For a complete exploration of this subject see [Pe88].

Suppose now that some variables $\{v_1, \ldots, v_l\}$ among $\{u_1, \ldots, u_n\}$ are assigned specific values $\{\mathbf{v}_1, \ldots, \mathbf{v}_l\}$, respectively. The *updating problem* is to compute the probability $P(u_i \mid v_1 = \mathbf{v}_1, \ldots, v_l = \mathbf{v}_l)$ for $i = 1, \ldots, n$. In principle, such computations are straightforward because each Bayesian network defines the joint probability distribution $P(u_1, \ldots, u_n)$ from which all conditional probabilities can be computed by dividing the appropriate sums. However, such computations are inefficient both in time and space unless they use conditional independence assumptions defined by (6). We shall see next how our approximation algorithms for the loop cutset problem reduce the computations needed for solving the updating problem.

A *trail* in a Bayesian network is a subgraph whose underlying graph is a simple path. A vertex b is called a *sink* with respect to a trail t if there exist two consecutive edges $a \to b$ and $b \leftarrow c$ on t. A trail t is *active by a set of vertices* Z if (1) every sink with respect to t either is in Z or has a descendant in Z and (2) every other vertex along t is outside Z. Otherwise, the trail is said to be *blocked* by Z.

Verma and Pearl [VePe88] have proved that if D is a Bayesian network of $P(u_1, \ldots, u_n)$ and all trails between a vertex in $\{r_1, \ldots, r_l\}$ and a vertex in $\{s_1, \ldots, s_k\}$ are blocked by $\{t_1, \ldots, t_m\}$, then the corresponding sets of variables $\{u_{r_1}, \ldots, u_{r_l}\}$ and $\{u_{s_1}, \ldots, u_{s_k}\}$ are independent conditioned on $\{u_{t_1}, \ldots, u_{t_m}\}$. Furthermore, Geiger and Pearl [GP90] proved a converse to this theorem. Both results are presented and extended in [GVP90].

Using the close relationship between blocked trails and conditional independence, Kim and Pearl [KiP83] developed an algorithm UPDATE-TREE that solves the updating problem on Bayesian networks in which every two vertices are connected with at most one trail. UPDATE-TREE views each vertex as a processor that repeatedly sends messages to each of its neighboring vertices. When equilibrium is reached, each vertex i contains the conditional probability distribution $P(u_i \mid v_1 = \mathbf{v}_1, \ldots, v_l = \mathbf{v}_l)$. The computations reach equilibrium regardless of the order of execution in time proportional to the length of the longest trail in the network.

Pearl [Pe86] solved the updating problem on any Bayesian network as follows. First, a set of vertices S is selected such that any two vertices in the network are connected by at most one *active* trail in $S \cup Z$, where Z is any subset of vertices. Then, UPDATE-TREE is applied once for each combination of value assignments to the variables corresponding to S, and, finally, the results are combined. This algorithm is called the method of *conditioning* and its complexity grows exponentially with the size of S. Note that according to the definition of active trails, the set S in Pearl's algorithm is a loop cutset of the Bayesian network. In this paper we have developed approximation algorithms for finding S.

When the domain size of the variables varies, then UPDATE-TREE is called a number of times which is bounded from above by the product of the domain sizes of the variables whose corresponding vertices participate in the loop cutset. If we take the logarithm of the domain size as the weight of a vertex, then solving the weighted loop cutset problem with these weights optimizes Pearl's updating algorithm in the case where the domain sizes are allowed to vary.

5. Discussion. It is useful to relate the feedback vertex set problem with the vertex cover problem in order to establish lower bounds on the performance ratios attainable for the feedback vertex set problem. A vertex cover of an undirected graph is a subset of the vertex set that intersects with each edge in the graph. The vertex cover problem is to find a minimum-weight vertex cover of a given graph. There is a simple polynomial reduction from the vertex cover problem to the feedback vertex set problem: Given a graph G, we extend G to a graph H by adding a vertex v_e for each edge $e \in E(G)$ and connecting v_e with the vertices in G with which e is incident in G. It is easy to verify that there always exists a minimum feedback vertex set in H whose vertices are all in $V(G)$ and this feedback vertex set is also a minimum vertex cover of G. In essence, this reduction replaces each edge in G with a cycle in H, thus transforming any vertex cover of G to a feedback vertex set of H.

Due to this reduction, it follows that the performance ratio obtainable for the feedback vertex set problem cannot be better than the one obtainable for the vertex cover problem. The latter problem has attracted a lot of attention over the years but has so far resisted any approximation algorithm that achieves in general graphs a constant performance ratio less than 2. We note that the above reduction retains planarity. However, for planar graphs, Baker [Bak94] provided a polynomial approximation scheme (PAS) for the vertex cover problem. For the UFVS problem, there are examples showing that 4 is the tightest constant performance ratio of algorithm SUBG-2-3.

Another consequence of the above reduction is a lower bound on the unweighted performance ratio of the greedy algorithm, GreedyCyc, for the feedback vertex set problem. In each iteration, GreedyCyc removes a vertex of maximal degree from the graph, adds it to the feedback vertex set, and removes all endpoints in the graph. A similar greedy algorithm for the vertex cover problem is presented in [Jo74] and in [Lo75]. The latter algorithm was shown to have an unweighted performance ratio no better than $\Omega(\log |V(G)|)$ [Jo74]. Due to the reduction to the cycle cover problem, the same lower bound holds also for GreedyCyc, as demonstrated by the graphs of [Jo74]. A tight upper bound on the worst-case performance ratio of GreedyCyc is unknown.

Finally, one should notice that the following heuristics may improve the performance ratios of our algorithms. For example, in each iteration MiniWCycle chooses to place in the cover all zero-weight vertices found on the smallest cycle. This choice might be rather poor especially if many weights are equal. It may be useful in this case to perturb the weights of the vertices before running the algorithm. Similarly, in algorithm SubG-2-3, there is no point in taking blindly all branchpoints of H. An appropriate heuristic here may be to pick the branchpoints one by one in decreasing order of residual degrees. Furthermore, the subgraph H itself should be constructed such that it contains as many high-degree vertices as possible.

Remark. In a preliminary version of this paper, presented in [BaGNR94], we conjectured that a constant performance ratio is attainable by a polynomial-time algorithm for the WFVS problem. This has been recently verified in [BaBF95, BeG96] where a performance ratio of 2 is obtained.

Acknowledgment. We would like to thank David Johnson for bringing [EP62] to our attention and Samir Khuller for helpful discussions.

REFERENCES

[BaBF95] V. Bafna, P. Berman, and T. Fujito, *Constant ratio approximations of the weighted feedback vertex set problem for undirected graphs*, in ISAAC 95, Algorithms and Computation, J. Staples, P. Eades, N. Katoh, and A. Moffat, eds., Lecture Notes in Computer Science 1004, Springer-Verlag, 1995, pp. 142–151.

[Bak94] B. S. Baker, *Approximation algorithms for NP-complete problems on planar graphs*, J. Assoc. Comput. Mach., 41 (1994), pp. 153–180.

[BaEv85] R. Bar-Yehuda and S. Even, *A local-ratio theorem for approximating the weighted vertex cover problem*, Ann. Discrete Math., 25 (1985), pp. 27–46.

[BaGNR94] R. Bar-Yehuda, D. Geiger, J. Naor, and R. M. Roth, *Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and Bayesian inference*, in Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, Arlington, VA, 1994, pp. 344–354.

[BeG96] A. Becker and D. Geiger, *Optimization of Pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem*, Artificial Intelligence, 83 (1996), pp. 167–188.

[DP87] R. Dechter and J. Pearl, *The cycle cutset method for improving search performance in AI*, in Proceedings 3rd IEEE on AI Applications, Orlando, FL, 1987.

[DP88] R. Dechter and J. Pearl, *Network-based heuristics for constraint satisfaction problems*, Artificial Intelligence, 34 (1988), pp. 1–38.

[De90] R. Dechter, *Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition*, Artificial Intelligence, 41 (1990), pp. 273–312.

[EP62] P. Erdős and L. Pósa, *On the maximal number of disjoint circuits of a graph*, Publ. Math Debrecen, 9 (1962), pp. 3–12.

[EP64] P. Erdős and L. Pósa, *On the independent circuits contained in a graph*, Canad. J. Math., 17 (1964), pp. 347–352.

[GJ79] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman, San Francisco, CA, 1979.

[GP90] D. Geiger and J. Pearl, *On the logic of causal models*, in Uncertainty in Artificial Intelligence 4, R.D. Shachter, T.S. Levitt, L.N. Kanal, and J.F. Lemmer, eds., North–Holland, New York, 1990, pp. 3–14.

[GVP90] D. Geiger, T. S. Verma, and J. Pearl, *Identifying independence in Bayesian networks*, Networks, 20 (1990), pp. 507–534.

[Ho82] D. S. Hochbaum, *Approximation algorithms for set covering and vertex covering problems*, SIAM J. Comput., 11 (1982), pp. 555–556.

[Ho83] D. S. Hochbaum, *Efficient bounds for the stable set, vertex cover, and set packing problems*, Discrete Appl. Math., 6 (1983), pp. 243–254.

[IR78] A. Itai and M. Rodeh, *Finding a minimum circuit in a graph*, SIAM J. Comput., 7 (1978), pp. 413–423.

[Jo74] D. S. Johnson, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci., 9 (1974), pp. 256–278.

[KhVY94] S. Khuller, U. Vishkin, and N. Young, *A primal-dual parallel approximation technique applied to weighted set and vertex cover*, J. Algorithms, 17 (1994), pp. 280–289.

[KiP83] H. Kim and J. Pearl, *A computational model for combined causal and diagnostic reasoning in inference systems*, in Proceedings of the Eighth IJCAI, Morgan–Kaufmann, San Mateo, CA, 1983, pp. 190–193.

[Lo75] L. Lovász, *On the ratio of optimal integral and fractional covers*, Discrete Math., 13 (1975), pp. 383–390.

[MS81] B. Monien and R. Schulz, *Four approximation algorithms for the feedback vertex set problem*, in Proceedings of the 7th Conference on Graphtheoretic Concepts of Computer Science, Hanser-Verlag, Munich, 1981, pp. 315–326.

[Pe86] J. Pearl, *Fusion, propagation and structuring in belief networks*, Artificial Intelligence, 29 (1986), pp. 241–288.

[Pe88] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan–Kaufmann, San Mateo, CA, 1988.

[Si67] M. Simonovits, *A new proof and generalizations of a theorem by Erdős and Pósa on graphs without $k+1$ independent circuits*, Acta Math. Acad. Hungaricae Tomus, 18 (1967), pp. 191–206.

[St90] J. Stillman, *On heuristics for finding loop cutsets in multiply connected belief networks*, in Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence, Cambridge, MA, 1990, pp. 265–272.

[SuC90] H. J. Suermondt and G. F. Cooper, *Probabilistic inference in multiply connected belief networks using loop cutsets*, Internat. J. Approx. Reason., 4 (1990), pp. 283–306.

[VePe88] T. Verma and J. Pearl, *Causal networks: Semantics and expressiveness*, in Proceedings of Fourth Workshop on Uncertainty in Artificial Intelligence, Minneapolis, MN, Association for Uncertainty in Artificial Intelligence, Mountain View, CA, 1988, pp. 352–359.

[Vo68] H. J. Voss, *Some properties of graphs containing k independent circuits*, in Proc. Colloq. Tihany, Academic Press, New York, 1968, pp. 321–334.

SIAM J. COMPUT.
Vol. 27, No. 4, pp. 960–971, August 1998
© 1998 Society for Industrial and Applied Mathematics
003

PLANAR INTEGER LINEAR PROGRAMMING IS NC EQUIVALENT TO EUCLIDEAN GCD*

D. F. SHALLCROSS†, V. Y. PAN‡, AND Y. LIN-KRIZ§

Abstract. It is not known if planar integer linear programming is P-complete or if it is in NC, and the same can be said about the computation of the remainder sequence of the Euclidean algorithm applied to two integers. However, both computations are NC equivalent. The latter computational problem was reduced in NC to the former one by Deng [*Mathematical Programming: Complexity and Application*, Ph.D. dissertation, Stanford University, Stanford, CA, 1989; *Proc. ACM Symp. on Parallel Algorithms and Architectures*, 1989, pp. 110–116]. We now prove the converse NC-reduction.

Key words. integer linear programming, Euclidean algorithm, greatest common divisor, parallel computational complexity

AMS subject classifications. 68Q15, 68Q22, 03D15

PII. S0097539794276841

1. Introduction.

Our main result is the proof of the NC-equivalence of the planar integer linear programming problem and the computation of the Euclidean remainder sequence, defined by the Euclidean algorithm for two integers, which terminates with their greatest common divisor (GCD). Hereafter, we will refer to these two major computational problems as 2-ILP and EUGCD, respectively. Both 2-ILP and EUGCD, as well as k-ILP, that is, the integer linear programming with k variables, for a fixed k, belong to the class P; specifically, sequential Boolean time $O(k^{9k}L\log L)$ suffices for solving k-ILP [7], [8], [12], and time $O(L^2)$ suffices for EUGCD [1], [11], where L denotes the input length (size).

On the other hand, very little is known about parallel complexity of both k-ILP and EUGCD. In particular, it is not known if either of these problems belongs to NC or if either of them is P-complete. (We recall that a computational problem is in NC if it can be solved by using $O((\log L)^c)$ time and $O(L^c)$ processors for a fixed c independent of the input length L; NC-reduction (or reduction in NC) of one problem to another is the reduction under the above bounds on time and on the processor number; NC-equivalence is the existence of NC-reductions in both directions; a P-complete problem is one whose solution in NC would imply the NC solution of all problems in P (see [4, Chapter 7], [5], [6], [10])).

There are relatively few computational problems for which, like for k-ILP and EUGCD, we neither know if they belong to NC nor if they are P-complete. Several computational problems of this kind are related to k-ILP and EUGCD via NC-reductions (see [2], [3], and Figure 1). These problems include GCD (computing the GCD of two integers), SGCD (solution of the set equation $m\mathbf{Z}+n\mathbf{Z}=d\mathbf{Z}$), expanding

*Received by the editors November 8, 1994; accepted for publication (in revised form) April 16, 1996; published electronically May 18, 1998. The results of this paper have been presented at the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '92) and at the 34th Annual IEEE Symposium on Foundations of Computer Science (FOCS '93).

http://www.siam.org/journals/sicomp/27-4/27684.html

†Bellcore, 445 South St., Morristown, NJ 07962 (davids@bellcore.com).

‡Department of Mathematics and Computer Science, Lehman College of the City University of New York, Bronx, NY 10468 (VPAN@lcvax.lehman.cuny.edu) and International Computer Science Institute, 1947 Center St., Berkeley, CA 94704. The work of this author was supported by NSF grant CCR-902690 and PSC CUNY awards 661340, 6622478, and 664334.

§1524 LSA Building, University of Michigan, Ann Arbor, MI 48109-1382 (ylinkriz@umich.edu).

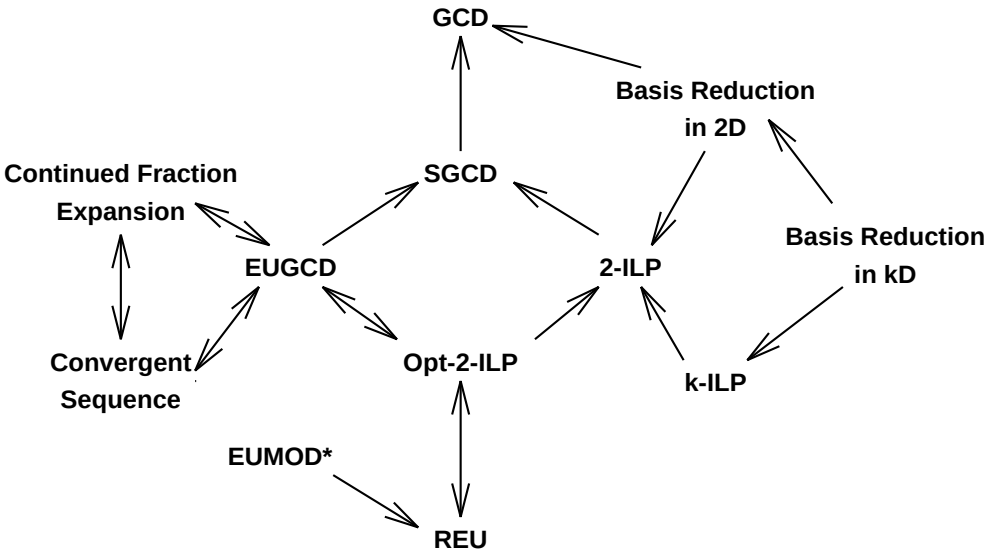


FIG. 1. *Arrowhead problems can be NC reduced to arrowtail problems.*

the continued fraction for the ratio of two integers, computing the sequence of their convergents, and reduction of a lattice of k dimensions for a fixed k.

Deng in [2], [3] proved NC-reduction from EUGCD to Opt-2-ILP, which is the (stronger) optimization version of 2-ILP (see Definition 1). (In Figure 1, we use 2-ILP for the feasibility integer linear programs.) The NC-reduction in the opposite direction, from Opt-2-ILP to EUGCD, turned out to be more elusive. Lin-Kriz and Pan in [13] established NC-equivalence between Opt-2-ILP and the problem they called REU, thus abbreviating relative EUGCD. REU amounts to solving both EUGCD and EUMOD*, the special case of MOD*, where MOD* denotes the problem of computing MOD*$(c, b_1, ..., b_n) = (...((c \bmod b_1) \bmod b_2)...) \bmod b_n$, for any set of natural numbers c, b_1, b_2, ..., b_n, whereas EUMOD* is the same problem where $b_1 > b_2$, $b_{i+2} = b_i \bmod b_{i+1}$, $i = 1, 2,...,n-2$; that is, in the input set for EUMOD*, the values b_3, ..., b_n have been generated as the successive remainders computed by the Euclidean algorithm for b_1 and b_2.

Although MOD* is a P-complete problem [9], we do not know if EUMOD* is in NC or if it is P-complete, so that the works [2], [3], and [13] still left establishing NC-equivalence between Opt-2-ILP and EUGCD as a research challenge. [13] proposed to try to reduce EUMOD* to EUGCD in NC, which would resolve the issue, but this has not worked so far.

The work on the version of [13] submitted for journal publication has brought in an additional coauthor (David Shallcross) and a stronger result, which ended up in the appearance of [14]. This paper combines the work of these two papers.

In this paper, we rely on applying a sequence of appropriate dissections and unimodular transformations of triangles, a method previously used in [16] to count the number of integer points in a polygon. We show that, used correctly, these techniques are powerful enough to arrive at the desired NC-reduction of Opt-2-ILP to EUGCD, which completes the long-awaited proof of NC-equivalence of these two problems.

2. Preliminaries. Hereafter, **Z**, **Q**, and **R**, as usual, denote the sets of integers, rationals, and reals, respectively. For (column) vectors a, b, c, ... from $\mathbf{Z}^2$ and $\mathbf{Q}^2$, we will let (a_1, a_2), (b_1, b_2), (c_1, c_2), ... denote the pairs of their coordinates. $\lfloor x \rfloor$ and $\lceil x \rceil$ denote the two integers closest to a rational x such that $\lfloor x \rfloor \le x \le \lceil x \rceil$.

EUGCD denotes the computational problem which requires, for an input pair a_0, a_1 of positive integers, to compute a sequence of integers a_2, a_3, ..., a_{n+2} such that

$$a_{i+2} = a_i - a_{i+1} \left\lfloor \frac{a_i}{a_{i+1}} \right\rfloor, \ i = 0, 1, ..., n, \qquad a_{n+1} > a_{n+2} = 0. \tag{1}$$

This is intimately related to continued fractions, as we can write

$$\frac{a_0}{a_1} = \left\lfloor \frac{a_0}{a_1} \right\rfloor + \frac{1}{\left\lfloor \frac{a_1}{a_2} \right\rfloor + \frac{1}{\lfloor \frac{a_2}{a_3} \rfloor + \cdots \frac{1}{a_n / a_{n+1}}}}. \tag{2}$$

Note that

$$n = O(\log a_0). \tag{3}$$

We will promiscuously mix representations of triangles by their vertices and by their facets or sides, because either representation can be transformed into the other in NC by performing some basic linear algebra operations.

Define a *unimodular* matrix to be an integer matrix with determinant either 1 or -1. Multiplying a unimodular matrix by an integer vector gives an integer vector, and since the inverse of a unimodular matrix is also unimodular, only an integer vector can multiply a unimodular matrix to give an integer vector.

DEFINITION 1. *An instance of Opt-2-ILP is the following: given $A \in \mathbf{Z}^{m \times 2}$, $v \in \mathbf{Z}^m$, $u = (u_1, u_2)^T \in \mathbf{Z}^2$, find $x = (x_1, x_2)^T \in \mathbf{Z}^2$ such that $Ax \le v$ that maximizes $u^T x = u_1 x_1 + u_2 x_2$, where v^T denotes the transpose of a vector v. That is, less formally, given a polygon P in $\mathbf{R}^2$ as an intersection of half-planes and a linear objective function, find the integer point that maximizes that function over all integer points in the polygon.*

We observe that a unimodular transformation of a triangle transforms a solution of Opt-2-ILP over this triangle to a solution of Opt-2-ILP over its image.

3. NC reduction of Opt-2-ILP to Opt-2-ILP over a triangle of a special form. The next two lemmas are from [13] (compare also [2], [3]); we clarify their proofs.

LEMMA 3.1. *Opt-2-ILP is NC-reducible to Opt-2-ILP over triangles.*

Proof. For an instance of Opt-2-ILP with feasible region P given in the usual manner as the intersection of half-planes, we may, in NC, compute the representation of P as a polygon given by vertices and edges. We will next show a relatively simple (although far from being the most efficient) method for doing this in NC. For every pair of half-planes compute the point where their boundary lines intersect (if they do intersect). For each such point, check whether it lies in each of the remaining half-planes and reject infeasible points. Eliminate duplicates. Finally, declare two points (now vertices) to be adjacent if they both lie on the boundary of the same half-plane. If unbounded P are allowed as input, we can add explicit bounds on the components of any finite optimal solution (see [15, Corollary 17.1c]).

We next triangulate this polygon in NC by taking an arbitrary vertex a, and, for each adjacent pair of vertices b, c, neither equal to a, producing the triangle abc. By

a note above, we can obtain in NC the equations of the sides of any triangle from its vertices. Now we solve the original optimization problem by, in parallel, solving the linear number of optimization problems over the triangles and taking the best of the optima. □

LEMMA 3.2. *Opt-2-ILP over triangles is NC-reducible to solving the following pair of computational problems*: *GCD and Opt-2-ILP over triangles of the form* $T =$*convex hull*(α, β, γ) *where* $\alpha = (\alpha_1, \alpha_2)^T$, $\beta = (\beta_1, \beta_2)^T$, $\gamma = (\gamma_1, \gamma_2)^T$ *are three points in* $\mathbf{Q}^2$ *such that* $\alpha_1 = \beta_1 < \gamma$, $\alpha_2 > \beta_2$, $\alpha_2 > \gamma$, *and* α *is the solution to the linear programming relaxation of the original Opt-2-ILP.*

Proof. We will rely on the proof of Lemma 5.2 of [13], which we will essentially reproduce with slightly changed notation for the triangles and their vertices and with some unnecessary claims removed.

We shall apply (unimodular) linear transformations

$$U \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} ax + by \\ cx + dy \end{pmatrix},$$

where $U = \left(\begin{smallmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{smallmatrix}\right)$ is a 2×2 matrix of the matrix class

$$GL_2(\mathbf{Z}) = \left\{ \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix} \ : \ g_{11}, g_{12}, g_{21}, g_{22} \in \mathbf{Z} \text{ and } g_{11}g_{22} - g_{21}g_{12} = \pm 1 \right\} .$$

Clearly, every such a transformation (with $U \in GL_2(\mathbf{Z})$) will map all the integral points into integral points; furthermore, if $U \in GL_2(\mathbf{Z})$, then $U^{-1} \in GL_2(\mathbf{Z})$, since

$$(\det\ U)U^{-1} = \begin{pmatrix} u_{22} & -u_{12} \\ -u_{21} & u_{11} \end{pmatrix}$$

and since

$$\det(U^{-1}) = \frac{u_{11}u_{22} - u_{12}u_{21}}{\det\ U} = 1 .$$

Now let A_0, B_0, C_0 denote the vertices of the triangle $T_0 = (A_0B_0C_0)$, which is the feasibility region of the input Opt-2-ILP. We now seek a linear transformation with the matrix $U \in GL_2(\mathbf{Z})$ that transforms the triangle $T_0 = (A_0B_0C_0)$ into a triangle $T = (ABC)$ satisfying the requirements of Lemma 3.2. Without loss of generality, we may assume that A_0 is an optimum for the LP relaxation of the input ILP and that all the angles of the triangle $T_0 = (A_0B_0C_0)$ are less than $\frac{\pi}{2}$ (Figure 2).

Next, we will transform the triangle $T_0 = (A_0B_0C_0)$ into a triangle $T = (ABC)$ so that $A - B$ will lie on the Y-axis and A (the image of A_0) will be the optimum point of correspondingly transformed linear program (see Figure 3). To formalize this stage, we let $A_0 - B_0 = \left(\begin{smallmatrix} z \\ t \end{smallmatrix}\right) = \vec{A_0B_0}$ denote the vector directed from B_0 to A_0, and we will seek a matrix

$$M = \begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix} \in GL_2(\mathbf{Z})$$

such that

$$\begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix} \begin{pmatrix} z \\ t \end{pmatrix} = \begin{pmatrix} 0 \\ v \end{pmatrix};$$

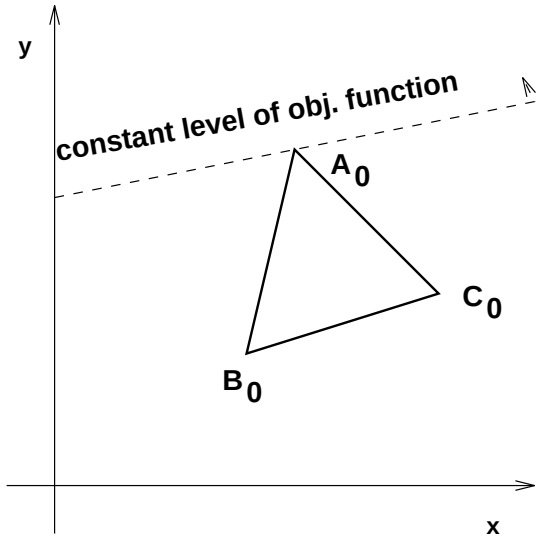


FIG. 2. *An arbitrary triangle* $A_0B_0C_0$.

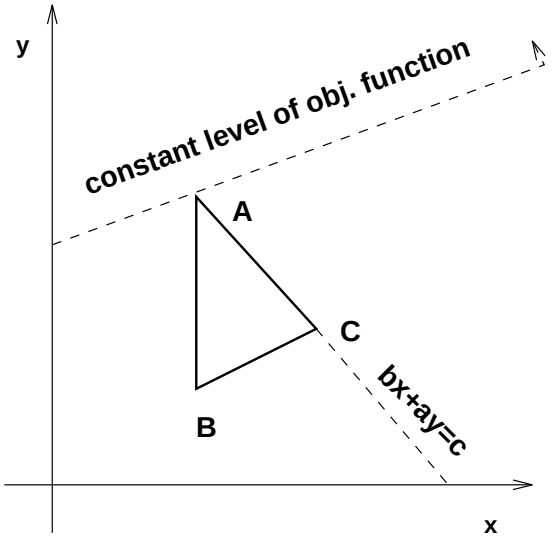


FIG. 3. *Triangle* $A_0B_0C_0$ *shifted to triangle* ABC.

that is,

$$m_{11}z + m_{12}t = 0 \ ,$$

$$m_{21}z + m_{22}t = v \ ,$$

$$m_{11}m_{22} - m_{12}m_{21} = \pm 1 \ .$$

Let $v = gcd(z,t)$. We can compute m_{21}, m_{22} by using the Euclidean algorithm or any algorithm that computes $v = gcd(z,t)$, since (as this can be immediately verified)

$$m_{11} = \frac{t}{v}\ , \quad m_{12} = -\ \frac{z}{v}\ ,$$

$$m_{11}m_{22} - m_{12}m_{21} = \frac{t}{v}\ m_{22} + \frac{z}{v}\ m_{21} = 1\ .$$

Let $T = (ABC)$ denote the resulting triangle.

During the entire transformation process, the image of A_0 (at the end represented by A'') remains the optimum of the correspondingly transformed linear program. Indeed, let $f(x, y) = ux + vy$ denote the objective function; then

$$ux + vy = \begin{pmatrix} u & v \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} u & v \end{pmatrix} M^{-1} M \begin{pmatrix} x \\ y \end{pmatrix},$$

and $M\binom{x}{y} = \binom{x'}{y'}$ is the optimum point in the new coordinate system, which is the image of the original optimum point. The vector $(u \quad v)M^{-1}$ is easy to calculate ($M^{-1} \in GL_2(\mathbf{Z})$ is the inverse of M), and it represents the direction of the constant level of the transformed objective function. Therefore, the optimum point and the optimum value are the transformation invariants.

Now we immediately observe that the requirements of Lemma 3.2 hold, except that we have yet to prove that $a > b$ in the equation for the edge $AC\ :\ bx + ay = c$ (Figure 3). Let k_{AC} denote the slope of AC; then all we need to prove is that $-\ \frac{\pi}{4} < k_{AC} \le 0$. We use the customary notation for the unimodular group of 2×2 matrices:

$$SL_2(\mathbf{Z}) = \left\{ \begin{pmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{pmatrix}\ :\ s_{11}s_{22} - s_{21}s_{12} = 1 \right\}\ .$$

Let $\alpha, \beta \in SL_2(\mathbf{Z})$ denote two such matrices:

$$\alpha = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}\ , \qquad \beta = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}\ ,$$

where $\alpha^{-1} = \beta$. Denote $\vec{CA} = C - A = \binom{g}{h}$; then

$$\text{(a)} \qquad \alpha \begin{pmatrix} g \\ h \end{pmatrix} = \begin{pmatrix} g \\ h - g \end{pmatrix}\ ,$$

$$\text{(b)} \qquad \beta \begin{pmatrix} g \\ h \end{pmatrix} = \begin{pmatrix} g \\ h + g \end{pmatrix}\ .$$

We can always assume that $g > 0$; otherwise we would have multiplied the resulting vector $\left[\begin{smallmatrix} g \\ h \end{smallmatrix}\right]$ by $\left(\begin{smallmatrix} -1 & 0 \\ 0 & 1 \end{smallmatrix}\right) \in GL_2(\mathbf{Z})$. If $h \ge g$, then we will multiply it by $\alpha^{\lfloor \frac{h}{g} \rfloor}$ and will arrive at the vector $\left[\begin{smallmatrix} g \\ h_1 \end{smallmatrix}\right]$, where $h_1 = h - g\lfloor \frac{h}{g} \rfloor < g$, as is required. Similarly, if $h < 0$, then we will multiply $\left[\begin{smallmatrix} g \\ h \end{smallmatrix}\right]$ by $\beta^{\lfloor \frac{-h}{g} \rfloor}$ and will arrive at $\left[\begin{smallmatrix} g \\ h_2 \end{smallmatrix}\right]$, where $h_2 = h + g\lfloor \frac{-h}{g} \rfloor > 0$. Thus, the requirements of Lemma 3.2 are satisfied for the triangle ABC. □

4. Restriction of the problem to the boundary of the convex hull of all the integer points of a superscribed right triangle. We will use the following definitions.

DEFINITION 2. *Hereafter, a triangle T of the form $\{(x_1, x_2) : ax_1 + bx_2 \le c, x_1 \ge g, x_2 \ge h\}$, for five integers $a > 0$, $b > 0$, c, g, h, will be called a* right triangle.

DEFINITION 3. *For any set S, let δS denote the boundary of the convex hull of the integer points in S.*

To solve Opt-2-ILP over a triangle of Lemma 3.2, we will reduce (in NC) this problem to solving EUGCD and to solving Opt-2-ILP over a few right triangles with integer slopes (see the next sections). To solve the latter problem in NC (see section 6), we will need the following lemma.

LEMMA 4.1. *Let a, b be positive integers, and let c, d, e, f, g be integers such that*

$$T = \{(x_1, x_2) : ax_1 + bx_2 \le c,\ dx_1 + ex_2 \le f,\ x_1 \ge g\}$$

is a nonempty bounded triangle. Let $u \in \mathbf{Z}^2$ be such that maximum over T of $u^T x$ occurs at the vertex $x^* = (g, (c - ag)/b)$. (This is the upper-left-hand corner of T.) Then the maximum over $T \cap \mathbf{Z}^2$ of $u^T x$ occurs among the integer points of $\delta T'$, the boundary of the convex hull H of integer points in the right triangle T', where

$$T' = \{(x_1, x_2) : ax_1 + bx_2 \le c,\ x_1 \ge g,\ x_2 \ge h\},$$

and h is an integer chosen so that $T \subset T'$.

Proof. If x^* defined above was an integer, then $x = x^*$ would optimize $u^T x$ over $T \cap \mathbf{Z}^2$ and would be a vertex on $\delta T'$. Otherwise let $x = z^*$ maximize $u^T x$ over $T \cap \mathbf{Z}^2$. Now suppose that $z^* \notin \delta T'$, but rather $z^* = (z_1^*, z_2^*) \in$ interior H, and we shall obtain a contradiction. (See Figure 4.)

Let K be the open wedge $\{(x_1, x_2) : x_1 < z_1^*,\ az_1^* + bz_2^* < ax_1 + bx_2\}$, and let P be the parallelogram $K \cap T'$. For all $x \in P$, $x_2 > z_2^*$, so P is in fact a subset of T. All points of K (and hence all of P) have a better objective value than z^*. In particular, by choice of z^*, P contains no integer points.

Since x^* is a vertex of T' but not an integer point, $x^* \notin H$. Thus since $z^* \in$ interior H, the diagonal of P from z^* to x^* intersects $\delta T'$ at some point t on an edge between two integer points w and y. Because they are integer points, neither w nor y can lie in P. They are in T', so they cannot lie anywhere else in K, but because the edge between them contains a point of P, one of these, say y, satisfies $y_1 < z_1^*$, $ay_1 + by_2 < az_1^* + bz_2^*$, and the other, w, satisfies $w_1 > z_1^*$, $aw_1 + bw_2 > az_1^* + bz_2^*$. Furthermore, either $w_2 > z_2^*$ or else $y_2 > z_2^*$.

Let $\bar{x} = w + y - z^*$. By the above, and since w and y are in T', we can see that $\bar{x}_1 > g$, and $a\bar{x}_1 + b\bar{x}_2 < c$. We can also see that $\bar{x}_2 > h$, so that $\bar{x} \in T'$. The points w and y are opposite vertices of a parallelogram with integer vertices y, $\bar{x}$, w, and z^* in T'. Since the point t between w and y lies on the boundary of the convex hull of integer points of T', we have the desired contradiction. Thus x^* lies on the boundary of H. ∎

5. Recursive partition of a triangle into unimodular images of right triangles.

LEMMA 5.1. *For integers k, l, $p > 0$, $q > 0$, r, the right triangle*

$$S = \left\{(x_1, x_2) : x_2 \le \frac{r}{q} - \frac{p}{q}x_1,\ x_1 \ge l,\ x_2 \ge k\right\}$$

(if nonempty) is the union (with disjoint interiors) of the right triangle

$$R = \left\{(x_1, x_2) : x_2 \le l' - \left\lfloor \frac{p}{q} \right\rfloor x_1,\ x_1 \ge l,\ x_2 \ge k\right\},$$

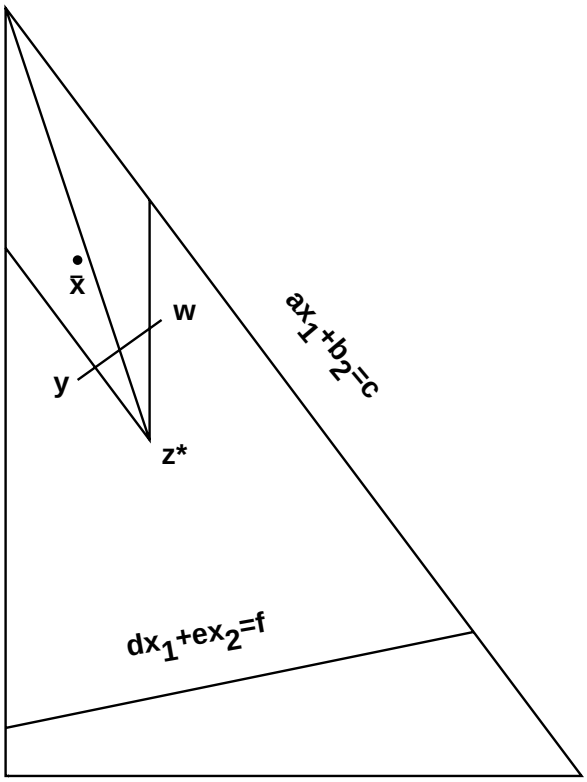


FIG. 4.

sharing the horizontal edge with the triangle S, and the unimodular image $U = MW$ of the triangle

$$W = \left\{ (x_1, x_2) : x_2 \le \frac{r}{q'} - \frac{p'}{q'} x_1,\ x_1 \ge l',\ x_2 \ge k' \right\},$$

where $p' = q$, $q' = p - q\lfloor \frac{p}{q} \rfloor$, $l' = k(\frac{q'}{p}) + \lfloor \frac{p}{q} \rfloor r/p$, $k' = l$, and

$$M = \begin{pmatrix} 0 & 1 \\ 1 & -\left\lfloor \frac{p}{q} \right\rfloor \end{pmatrix}.$$

Proof. We just divide S along the line of slope $-\lfloor \frac{p}{q} \rfloor$ that goes through the point $(\frac{r}{p} - (\frac{q}{p})k, k)$, the lower-right-hand corner of S. (See Figure 5.) This gives $S = R \cup U$ where R is as above, and

$$U = \left\{ (x_1, x_2) : l' - \left\lfloor \frac{p}{q} \right\rfloor x_1 \le x_2 \le \frac{r}{q} - \frac{p}{q} x_1,\ x_1 \ge l \right\}.$$

Now perform an elementary unimodular transformation $(x_1, x_2) \to (x_1, \lfloor \frac{p}{q} \rfloor x_1 + x_2)$ to map U to the triangle

$$V = \left\{ (x_1, x_2) : l' \le x_2 \le \frac{r}{q} - \left(\frac{p}{q} - \left\lfloor \frac{p}{q} \right\rfloor \right) x_1,\ x_1 \ge l \right\}.$$

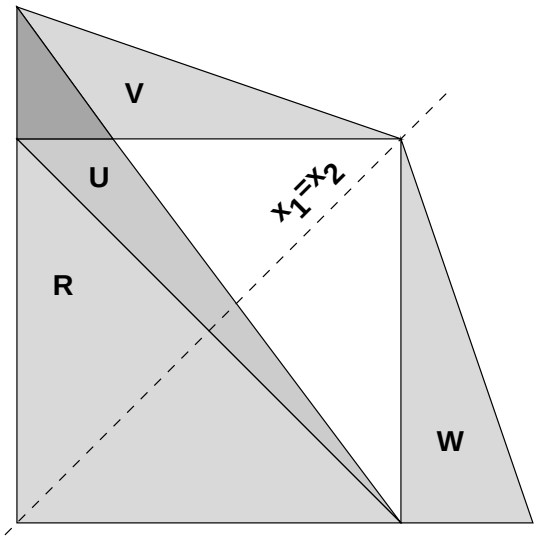


FIG. 5. *Lemma* 5.1 *for* $k = l = 0$.

Notice that we have $(\frac{p}{q}) - \lfloor \frac{p}{q} \rfloor = \frac{q'}{p'}$. Reflecting V over the line $x_1 = x_2$ gives us W above. □

LEMMA 5.2. *Given a sequence* $a_0, a_1, \ldots, a_{n+1}$ *of integers satisfying* (1) *and three rationals* h, k_0, *and* l_0 *such that* $a_0 l_0 + a_1 k_0 \leq h$, *we can compute in NC three sequences of rationals* k_i, l_i, *and unimodular matrices* M_i *such that if*

$$T^* = \{(x_1, x_2) : a_0 x_1 + a_1 x_2 \leq h,\ x_1 \geq l_0,\ x_2 \geq k_0\}$$

and

$$V_i = \left\{ (x_1, x_2) : x_2 \leq l_{i+1} - \left\lfloor \frac{a_i}{a_{i+1}} \right\rfloor x_1,\ x_1 \geq l_i,\ x_2 \geq k_i \right\},$$

then

$$T^* = \bigcup_{i=0}^{n} M_i V_i.$$

Proof. First we give recursions for the sequences k_i, l_i, and M_i. Then we will show that these sequences actually meet the requirements of the lemma.

Let

$$M_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

and, for $0 \leq i < n$,

$$k_{i+1} = l_i, \tag{4}$$

$$l_{i+1} = k_i \frac{a_{i+2}}{a_i} + \left\lfloor \frac{a_i}{a_{i+1}} \right\rfloor \frac{h}{a_i}, \tag{5}$$

$$\tilde{M}_i = \begin{pmatrix} 0 & 1 \\ 1 & -\left\lfloor \frac{a_i}{a_{i+1}} \right\rfloor \end{pmatrix}, \tag{6}$$

$$M_{i+1} = M_i \tilde{M}_i. \tag{7}$$

Due to (3)–(7), the maximum magnitude μ of l_i and of the entries of M_i (over all i) satisfies the relation

$$\log \mu = (\log(a_0 + a_1))^{O(1)}. \tag{8}$$

For $0 \le i \le n$, define the triangle

$$S_i = \left\{ (x_1, x_2) : x_2 \le \frac{h}{a_{i+1}} - \frac{a_i}{a_{i+1}} x_1,\ x_1 \ge l_i, x_2 \ge k_i \right\},$$

so $S_0 = T^*$. Applying Lemma 5.1, for $S = S_i$, $p = a_i$, $q = a_{i+1}$, $r = h$, $V = V_i$, $l = l_i$, $k = k_i$, $l' = l_{i+1}$, $W = S_{i+1}$, $p' = a_{i+1}$, $q' = a_{i+2}$, $M = \tilde{M}_i$, we can see that S_i is the union of the two triangles (with disjoint interiors) V_i and $\tilde{M}_i S_{i+1}$, where $S_{n+1} = \emptyset$. By induction, we easily deduce that $T = \bigcup_{i=0}^{n-1} M_i V_i$.

Now, due to (8), and since the values k_i and l_i are defined by a linear recurrence with coefficients depending only on h and the a_i, we may compute these values k_i and l_i in NC using the prefix algorithm on their transformation matrix. Similarly, the prefix algorithm allows one to compute in NC the M_i as the products of the $\tilde{M}_i$. □

Remark. If we let $u_i = (h - a_{i+1}k_i)/a_i$, then it can be verified, for $i \ge 2$, that

$$l_i = \left\lfloor \frac{a_{i-1}}{a_i} \right\rfloor u_{i-1} + l_{i-2},$$

$$u_i = \left\lfloor \frac{a_{i-1}}{a_i} \right\rfloor l_{i-1} + u_{i-2}.$$

Thus, if h, k_0, and l_0 are integers, u_0 and l_1 can be expressed as fractions with denominator a_0, and all later terms can be expressed as fractions with denominator $a_0 a_1$. Likewise, if h, k_0, and l_0 are fractions with denominators d_h, d_k, and d_l, all terms can be expressed as fractions with denominator $d_h d_k d_l a_0 a_1$.

6. Final reduction of Opt-2-ILP to EUGCD.

LEMMA 6.1. *Opt-2-ILP over a triangle of the form T = convex hull(α, β, γ), where α, β, and γ are points in $\mathbf{Q}^2$ such that $\alpha_1 = \beta_1 < \gamma_1$, $\alpha_2 > \beta_2$, and $\alpha_2 \ge \gamma_2$, and with objective vector $u \in \mathbf{Z}^2$ such that $u^T\alpha > u^T\beta$ and $u^T\alpha > u^T\gamma$, can be NC-reduced to EUGCD.*

Proof. Given such a triangle by its vertices, we easily convert it in NC to the representation used in the statement of Lemma 4.1 for T. We then trivially produce the parameters for the triangle T' in the statement of that lemma. Using one call to EUGCD, we produce the sequence $a_0, a_1, \ldots, a_n, a_{n+1}$ according to the relation (1), loosely speaking, the continued fraction representation of the slope a_0/a_1 of T'. We now apply Lemma 5.2 to T', computing in NC the unimodular matrices M_i and right triangles V_i with integer slopes such that $T' = \cup_i M_i V_i$. According to Lemma 4.1, the maximum of $u^T x$ over $x \in T$ occurs in $\delta T'$ and so in $T \cap \delta T'$. The set $\delta T'$ is contained in $\cup_i \delta(M_i V_i)$, so $T \cap \delta T'$ is contained in $T \cap \cup_i \delta(M_i V_i)$, which equals $\cup_i (T \cap M_i \delta V_i)$.

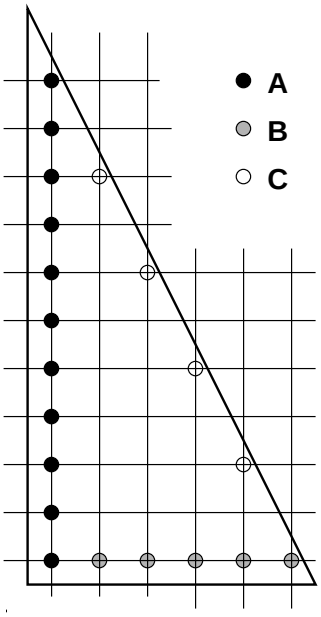


FIG. 6.

Each δV_i can be expressed simply as the union of three pieces A_i, B_i, C_i, each consisting of an arithmetic sequence of integer points, as follows:

$$\delta V_i = A_i \cup B_i \cup C_i,$$

$$A_i = \left\{ (\lceil l_i \rceil, \lceil k_i \rceil + j), \right.$$
$$\left. j = 0, \ldots, \lfloor l_{i+1} \rfloor - \left\lfloor \frac{a_{i+1}}{a_i} \right\rfloor \lceil k_i \rceil - \lceil k_i \rceil \right\},$$

$$B_i = \left\{ (\lceil l_i \rceil + j, \lceil k_i \rceil), \right.$$
$$\left. j = 0, \ldots, \frac{\lfloor l_{i+1} \rfloor - \lceil k_i \rceil}{\left\lfloor \frac{a_{i+1}}{a_i} \right\rfloor} - \lceil l_i \rceil \right\},$$

$$C_i = \left\{ \left(\lceil l_i \rceil + j, \lfloor l_{i+1} \rfloor - \left\lfloor \frac{a_{i+1}}{a_i} \right\rfloor \lceil l_i \rceil - \left\lfloor \frac{a_{i+1}}{a_i} \right\rfloor j \right), \right.$$
$$\left. j = 0, \ldots, \frac{\lfloor l_{i+1} \rfloor - \lceil k_i \rceil}{\left\lfloor \frac{a_{i+1}}{a_i} \right\rfloor} - \lceil l_i \rceil \right\}.$$

(See Figure 6.) Naturally, $M_i \delta V_i$ is the union of $M_i A_i$, $M_i B_i$, and $M_i C_i$. Thus we can maximize $u^T x$ over $x \in T$ by taking the maximum (for $i = 0, 1, ..., n$) of the maxima of $u^T x$ over x in $T \cap M_i A_i$, in $T \cap M_i B_i$, and in $T \cap M_i C_i$. □

THEOREM 6.2. *Opt-2-ILP can be NC-reduced to EUGCD.*

Proof. By Lemma 3.1, Opt-2-ILP can be NC-reduced to Opt-2-ILP over triangles. By Lemma 3.2, Opt-2-ILP over triangles can be NC-reduced to EUGCD plus Opt-2-ILP over triangles of a particular form. Finally, by Lemma 6.1 Opt-2-ILP can be NC-reduced to EUGCD. □

Since [2], [3] give a reduction of EUGCD to Opt-2-ILP, we obtain that Opt-2-ILP and EUGCD are NC-equivalent.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison–Wesley, Reading, MA, 1974.

[2] X. DENG, *Mathematical Programming: Complexity and Application*, Ph.D. dissertation, Stanford University, Stanford, CA, 1989.

[3] X. DENG, *On parallel complexity of integer linear programming*, in Proc. ACM Symp. on Parallel Algorithms and Architectures, Association for Computing Machinery, New York, 1989, pp. 110–116.

[4] A. GIBBONS, W. RYTTER, *Efficient Parallel Algorithms*, Cambridge University Press, London, 1988.

[5] R. GREENLAW, H. J. HOOVER, AND W. L. RUZZO, *A compendium of problems complete for P*, Tech. report TR 91-11, Dept. of Computer Science, University of Alberta, Edmonton; Tech. report TR 99-05-01, CS and Engineering Dept., University of Washington, Seattle, 1991.

[6] R. GREENLAW, *Polynomial completeness and parallel computation*, in Synthesis of Parallel Algorithms J. H. Reif, Ed., Morgan-Kaufman Publishers, San Mateo, CA, 1993, pp. 901–954.

[7] R. KANNAN, *A polynomial algorithm for the two-variable integer linear programming problem*, J. Assoc. Comput. Mach., 27 (1980), pp. 118–122.

[8] R. KANNAN, *Improved algorithms for integer linear programming and related lattice problems*, in Proc. 15th Annual ACM Symp. on Theory of Computing, Association for Computing Machinery, New York, 1983, pp. 193–206.

[9] H. H. KARLOFF AND W. L. RUZZO, *The complexity of iterated MOD function*, Inform. Comput., 80 (1989), pp. 193–204.

[10] R. M. KARP AND V. RAMACHANDRAN, *A survey of parallel algorithms for shared-memory machines*, in Handbook of Theoretical Computer Science, North–Holland, Amsterdam, 1990, pp. 869–941.

[11] D. E. KNUTH, *The Art of Computer Programming Vol.* 2: *Seminumerical Algorithms*, Addison–Wesley, Reading, MA, 1981.

[12] H. W. LENSTRA, JR., *Integer linear programming with a fixed number of variables*, Math. Oper. Res., 8 (1983), pp. 538–548.

[13] YU LIN-KRIZ AND V. Y. PAN, *On parallel complexity of integer linear programming, gcd, and the iterated mod function*, in Proc. 3rd Ann. ACM-SIAM Symp. on Discrete Algorithms, 1992, pp. 124–137.

[14] D. F. SHALLCROSS, V. Y. PAN, AND YU LIN-KRIZ, *The NC equivalence of planar integer linear programming and Euclidean GCD*, in Proc. 34th Ann. Symp. on Foundations of Computer Science, IEEE Computer Soc. Press, Los Alamitos, CA, 1993, pp. 557–564.

[15] A. SCHRIJVER, *Theory of Linear and Integer Programming*, John Wiley and Sons, New York, 1986.

[16] L. YA. ZAMANSKII AND V. L. CHERKASSKII, *A formula for finding the number of integer points under a straight line and its application*, Èkonom. i Mat. Metody, 20 (1984), pp. 1132–1138 (in Russian).

SIAM J. COMPUT.
Vol. 27, No. 4, pp. 972–992, August 1998
© 1998 Society for Industrial and Applied Mathematics
004

ALL HIGHEST SCORING PATHS IN WEIGHTED GRID GRAPHS AND THEIR APPLICATION TO FINDING ALL APPROXIMATE REPEATS IN STRINGS*

JEANETTE P. SCHMIDT†

Abstract. Weighted paths in directed grid graphs of dimension $(m \times n)$ can be used to model the string edit problem, which consists of obtaining optimal (weighted) alignments between substrings of A, $|A| = m$, and substrings of B, $|B| = n$. We build a data structure (in $O(mn \log m)$ time) that supports $O(\log m)$ time queries about the weight of any of the $O(m^2 n)$ best paths from the vertices in column 0 of the graph to all other vertices. Using these techniques we present a simple $O(n^2 \log n)$ time and $\Theta(n^2)$ space algorithm to find all (the locally optimal) approximate tandem (or nontandem) repeats xy within a string of size n. This improves (by a factor of $\log n$) upon several previous algorithms for this problem and is the first algorithm to find *all locally optimal* repeats. For edit graphs with weights in $\{0, -1, 1\}$, a slight modification of our techniques yields an $O(n^2)$ algorithm for the cyclic string comparison problem, as compared to $O(n^2 \log n)$ for the case of general weights.

Key words. string matching, tandem repeats, edit-distance, dynamic programming

AMS subject classification. 68P99

PII. S0097539795288489

1. Introduction.

An approximately repeated pattern in a string A is a substring x followed by a substring y, so that x and y match approximately. If x and y are adjacent in A the repeat is called a tandem repeat. An important motivation for the *approximate* tandem repeat problem is found in molecular biology. From a computational point of view, DNA can be viewed as a string over a four letter alphabet (the four nucleotides), while a protein can be viewed as a string over a twenty letter alphabet (the twenty amino acids). Adjacent repeated patterns, or tandem repeats, occur frequently and play an important role in both DNA and protein sequences. Although the function of the repeating patterns that appear on the DNA or protein level is not always well understood, they have been associated with several important properties. For example, researchers have recently discovered that the tips of chromosomes are made up of telomeres, consisting of many repetitions of the same pattern. Each time the cell divides a fixed number of such repetitions is lost. The number of remaining repetitions appears to hold the cell's clock and to determine the number of divisions the cell can still undergo. On the protein level repetitions have been associated with structural properties and have helped determine the secondary structure of proteins, that is, its local folding pattern. As is inevitably the case in biological applications, the repetitions occurring in both DNA and protein sequences are not exact. It is therefore natural to look for approximately repeated patterns.

A very general measure of an approximate match between two strings can be expressed as a shortest (or longest) path in weighted grid graphs. An (m, n) grid graph $G = (V, E)$ is a directed (acyclic) weighted graph whose vertices are the $(m+1) \times (n+1)$

*Received by the editors July 5, 1995; accepted for publication (in revised form) April 30, 1996; published electronically May 18, 1998.

http://www.siam.org/journals/sicomp/27-4/28848.html

†Department of Computer Science, Polytechnic University, 6 MetroTech, Brooklyn, NY 11201 (jps@pucs4.poly.edu). This author was partially supported by NSF grants CCR-9305873 and HRD-9627109 and the New York State Science and Technology Foundation Center for Advanced Technology.

points of the grid with rows $0 \ldots m$ and columns $0 \ldots n$. Vertex (i, j) has a directed edge to $(i+1, j)$, $(i+1, j+1)$, and $(i, j+1)$, provided that these endpoints are within the boundaries of the grid (Figure 1).

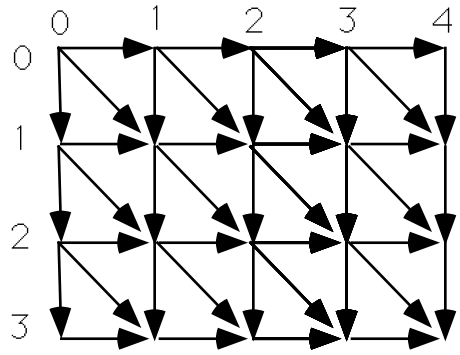


FIG. 1. *A* 3×4 *grid graph.*

The string edit problem is modeled by a grid graph and consists of obtaining an optimal alignment between a substring A' of $A = A_1 \ldots A_m$ and B' of $B = B_1 \ldots B_n$ (i.e., transforming A' into B' by performing a "least cost" series of weighted edit operations on A'). An alignment of $A_i^c = A_{i+1} \ldots A_c$ with $B_{i'}^{c'} = B_{i'+1} \ldots B_{c'}$ corresponds to a path from (i, i') to (c, c') in the graph. An edge $\langle (k, l), (k+1, l) \rangle$ (resp., $\langle (k, l), (k, l+1) \rangle$) on this path with weight w corresponds to skipping over character A_{k+1} (resp., character B_{l+1}), which incurs a penalty of w, while edge $((k, l), (k+1, l+1))$ with weight w' corresponds to aligning A_{k+1} and B_{l+1} with penalty (reward) w', which can be either positive or negative. The weight w of a given edge can be specified directly in the grid graph, or as is frequently the case in biological applications, is given by a penalty matrix (which specifies the substitution cost for each pair of characters and the deletion cost for each character from the alphabet). Depending on whether a positive weight corresponds to a penalty or a reward a best alignment of A_i^c with $B_{i'}^{c'}$ is given by a shortest (resp., longest) weighted path from (i, i') to (c, c') in the grid graph. Since edges can assume negative weights in either case, finding the longest or shortest path is analogous. We will henceforth assume that best alignments correspond to longest (i.e., highest scoring) paths. Computing such paths and hence a best alignment with arbitrary substitution/deletion weights can be done by standard dynamic programming in $O(|A_i^c||B_{i'}^{c'}|)$ time. If weights are arbitrary this is also optimal [H-88].

When trying to identify repetitions in a string we may initially seek the pair $(A_i^c, A_{i'}^{c'})$ whose score is highest among all such pairs. It turns out that under standard similarity measures it appears to be much easier to find "the highest scoring" repeated pattern when the patterns, A_i^c and $A_{i'}^{c'}$, are allowed to overlap (i.e., when c' is not required to be greater than or equal to i). This is discussed in more detail in section 2. In many applications, however, including the ones mentioned earlier, one is really interested in finding nonoverlapping repeated patterns. This problem was addressed in [M-92], [KM-93], and [B-94] using a general penalty matrix, and, among others, in [ML-84], [AP-83], and [C-81] for exact repetitions. In [KM-93] a recursive algorithm is described that reports "the highest scoring" nonoverlapping approximate repeat in $O(n^2 \log^2 n)$ time and $O(n^2 \log n)$ space. A modification of their algorithm was given by [B-94], which reduced the space to $O(n^2)$. [LS-93] study the problem of finding

good tandem repeats under the Levenshtein distance, in which matching characters score 0 and deletions/substitutions of characters score 1, and report repeated patterns whose edit distance is less than k in time $O(kn \log k \log (n/k))$. (The complexity stated in [LS-93] is actually $O(kn \log k \log n)$, but a more careful analysis of the algorithm given there gives the slightly stronger bound stated above.) The algorithm presented in this article is designed for arbitrary penalty matrices, runs in $O(n^2 \log n)$ time, and uses $O(n^2)$ space. Another major advantage of our algorithm over [KM-93], [B-94], in addition to its improved running time, is that it reports all "locally optimal" repeats, and not just the highest scoring pair. This is formally defined in section 3.

The remainder of this article is organized as follows. In section 2 we address the problem of finding highest scoring paths in grid graphs and introduce the notion of *locally optimal* alignments (and paths), as defined in [ES-83]. In section 3 we show how to efficiently compute the highest scoring paths from all points on column zero to all other points in the grid graph. In section 4 we show how the techniques developed in sections 2 and 3 can be applied to solve the approximate *tandem* repeat problem. In section 5 we show how the algorithm can be modified to report locally optimal nontandem repeats. Finally, in section 6 we discuss an application to grid graphs with unit weights.

2. High scoring paths in grid graphs. In this section we review some standard string matching problems and the (dynamic programming) techniques for solving them. We also define the notion of locally optimal alignments and give a general framework for obtaining such alignments. Finally, we show how the notion of locally optimal alignments applies to approximate repeats in a string, and we give an example to explain why the "trivial" solution for identifying repetitions does not give the desired results.

Finding the best alignment between a string A and a string B is one of many string editing problems that occur in a number of applications. To solve this problem alignments between substrings of A and substrings of B are examined. Let $score(A_j^r, B_\ell^t)$ denote the score of the best alignment between $A_j^r = A_{j+1} \dots A_r$ and $B_\ell^t = B_{\ell+1} \dots B_t$. The standard dynamic programming solution for finding $score(A, B)$ defines $T[r, t]$ as $score(A_0^r, B_0^t)$ and computes $T[r, t]$ from the T values of the points in $\{(r-1, t), (r-1, t-1), (r, t-1)\}$ which are contained in the grid. $T[0, 0]$ is initialized to zero, and the solution, $score(A, B)$, is given in $T[m, n]$. Another common problem consists of seeking the substring of B that best matches A. This corresponds to finding a highest scoring path from row zero to row m in the grid. The only difference in the dynamic programming is that now $T[r, t]$ holds the maximum in $\{score(A_0^r, B_\ell^t) \mid \ell \in [0, t]\}$. The computation of $T[r, t]$, for $r > 0$, given the T values of the relevant points in $\{(r-1, t), (r-1, t-1), (r, t-1)\}$, remains the same. The change in the computation is reflected in the initial assignments $T[0, t] = 0$, for $t \in [0, m]$ (assuming as usual that horizontal edges in the graph carry negative weights). The solution $\max_{0 \le \ell \le t \le m} score(A, B_\ell^t)$ is given in $\max_{t \in [0,m]} T[m, t]$.

A somewhat harder problem occurs when no restrictions on the substrings from A and B are given and one seeks to find substrings of A which align well with substrings of B. This latter problem is called the local alignment problem, since "local" pieces of the two strings are matched. Provided that the score of the alignment of two empty strings is defined as 0, and only pairs whose alignment scores above 0 are of interest, [SW-81] showed that essentially the same dynamic $O(n^2)$ programming solution can still be used. $T[r, t]$ would now hold the maximum in $\{score(A_j^r, B_\ell^t) \mid j \in [0, r], \ell \in [0, t]\}$. Note that if none of the suffixes of A_0^r and B_0^t can be aligned with a strictly

positive score, $T[r,t]$ would be set to $score(A_r^r, B_t^t) = 0$. The score for the highest scoring pair is given in $\max_{r\in[0,m],t\in[0,n]} T[r,t]$, but generally this is not the only pair of interest. The question of what other pairs of matching substrings from A and B to report, or in other words what pairs of substrings should be considered locally optimal, has been analyzed in [ES-83] and [WE-87], among others. We will adopt the definition of local optimality as given in Erickson and Sellers [ES-83], which we now describe.

DEFINITION 2.1. *The pair (A_j^r, B_ℓ^t) is locally optimal and satisfies the threshold if the following three conditions hold.*

1. $score(A_j^r, B_\ell^t) \geq$ *threshold;*
2. $\forall r' \geq j$ *and* $t' \geq \ell$, $\quad score(A_j^r, B_\ell^t) \geq score(A_j^{r'}, B_\ell^{t'})$;
3. $\forall j' \leq r$ *and* $\ell' \leq t$, $\quad score(A_{j'}^r, B_{\ell'}^t) \leq score(A_j^r, B_\ell^t)$.

Locally optimal alignments with any threshold $\tau > 0$ can be found in two passes of dynamic programming, each pass as just described. In the first pass each endpoint (r,t) records a pair of indexes (j,ℓ) which maximizes $score(A_j^r, B_\ell^t)$, by setting $S[r,t] = (j,\ell)$. This is easily done during the computation of $T[r,\ell]$, since (j,ℓ) satisfies $score(A_j^r, B_\ell^t) = T[r,t]$. The second pass is a (backward) dynamic programming on the transposed grid graph, where values $T^b[j,\ell]$ hold the maximum in $\{score(A_j^r, B_\ell^t) \mid r \in [j,m], t \in [\ell,n]\}$. Each point (j,ℓ) records a pair of indexes (r,t) which maximizes $score(A_j^r, B_\ell^t)$ $(r \geq j, t \geq \ell)$ by setting $E[j,\ell] = (r,t)$. It is not difficult to verify that if the above maxima are unique, a pair (A_j^r, B_ℓ^t) is *locally optimal* (according to Definition 2.1) if and only if $S[r,t] = (j,\ell)$ and $E[j,\ell] = (r,t)$. After the two dynamic programming passes, the locally optimal pairs and (with some extra standard bookkeeping) their alignments can be reported. When the suffixes of A_0^r and A_0^t (resp., prefixes of A_j^m and B_ℓ^n) yielding the best alignment are not unique one can either use a tie-breaking rule (such as the maximum j and maximum ℓ) or record all pairs (j,ℓ) (resp., (r,t)) yielding the highest score. In the latter case $S[r,t]$ and $E[j,\ell]$ hold a set of points. A pair (A_j^r, B_ℓ^t) would be considered locally optimal if $(j,\ell) \in S[r,t]$ and $(r,t) \in E[j,\ell]$.

Similar questions, as just described, are of considerable interest when comparing the string A to itself. Due to the symmetric nature of the grid graph corresponding to the comparison of a string with itself, the analysis can be limited to the upper triangular grid graph. In addition, the diagonal of the grid graph is initialized to zero, to exclude the possibility of the uninteresting case corresponding to any character A_j matching itself in an alignment.

Consider hence the (upper triangular) $(n+1)\times(n+1)$ grid graph obtained when comparing string $A = A_1, \ldots, A_n$ with itself. An approximate repeat (A_j^r, A_ℓ^t) with $\ell \geq r$ corresponds to the highest scoring path in the (upper triangular) grid graph from point (j,ℓ) to point (r,t). If $score(A_j^r, A_\ell^t)$ is above the (predetermined) threshold, the pair (A_j^r, A_ℓ^t) would be considered an approximate repeat in A. If $\ell = r$, so that there is no gap between the two substrings, the repeat is tandem. All approximate *tandem* repeats (A_j^c, A_c^t) correspond to a path in the (upper triangular) grid graph from a point on column c to a point on row c. If pairs (A_j^r, A_ℓ^t) are included for which $\ell < r$ (indicating that the strings overlap), the problem changes drastically. Finding the best scoring (possibly overlapping) repeat reduces to finding the substrings A_j^r and A_ℓ^t, for which the weight of the optimal path from (j,ℓ) to (r,t) in the (upper triangular) grid graph is highest among all such pairs, or alternatively finding the pairs of substrings whose alignments are locally optimal . The above problem can be directly solved by (one or two passes of) dynamic programming on a triangular grid graph,

as just described for square matrices. One problem in including these overlapping repeats is that in many applications nonoverlapping repeats are sought. In addition, these overlapping repeats present an "unfair competition" in the search for the best repeats. The best tandem repeat (or highest scoring path from some column c to some row c) might be very different than the best path starting at an earlier column c'. More precisely, the overlapping repeats may "hide" interesting tandem repeats. Consider, for example, the string $A = IRQIQLWLRQIWIRLRQL$, compared to itself. Figure 2 shows a penalty matrix and the resulting (upper triangular) dynamic programming matrix of A versus A. With the specified penalty matrix the (highest scoring) tandem repeat $IRQI\lfloor QLWLR\rfloor\lceil QIWIR\rceil LRQL$ would not be found. The alignments that would be found (given a threshold value 3) would be the (approximate) nontandem repeat $\lfloor IRQI\rfloor QLWLRQIWIR\lceil LRQL\rceil$, the (approximate) nontandem repeat $\lfloor \mathbf{IR}Q\mathbf{IQL}\rfloor WLRQIW\lceil \mathbf{IRL}R\mathbf{QL}\rceil$, and the long overlapping repeat $\lfloor IRQIQLW\lceil LRQI\rfloor WIRLRQL\rceil$, whose alignments are shown in Figure 2.

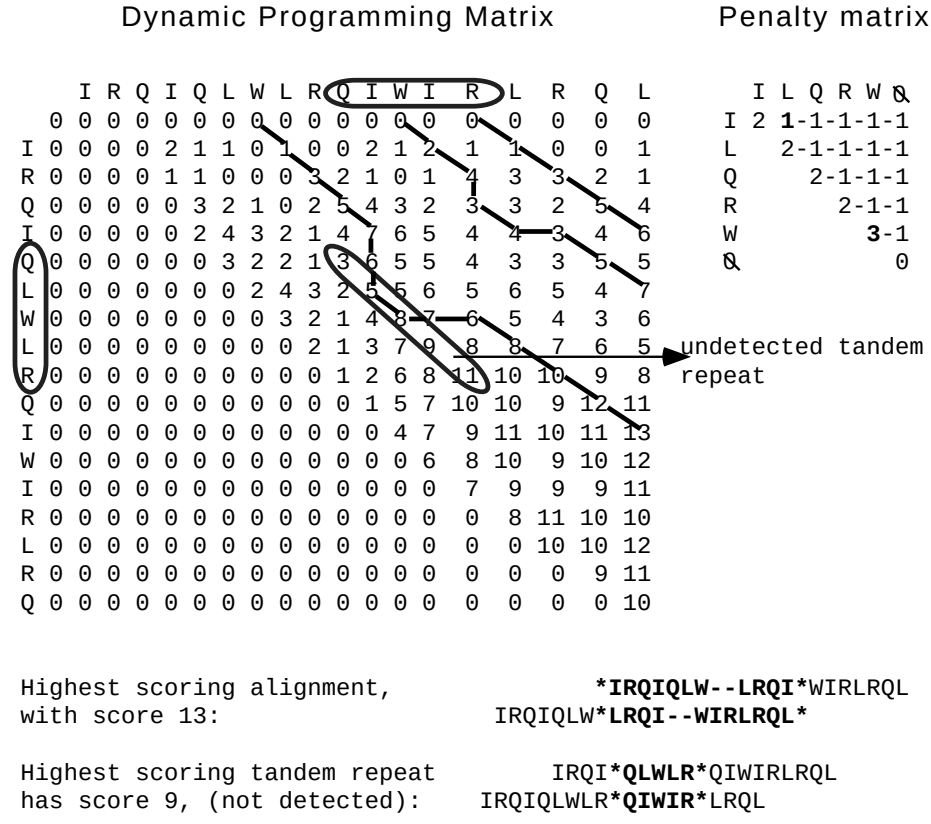


FIG. 2. *Example of a tandem repeat, not detected in a global search.*

It is worth noting that overlapping repeats are of independent interest, as they might indicate that a smaller string is (approximately) repeated several times. It appears that a different scoring method might be needed to detect both many repetitions of the same string and nonoverlapping repeats in the same search.

Note that a simple (but expensive) solution to the problem of finding nonover-

lapping repeats is to compare A_0^c with A_c^n, for $c = 1 \ldots n$. The comparison of A_0^c and A_c^n is modeled by a (rectangular) grid graph G_c of size $(c+1) \times (n-c+1)$. For nonoverlapping, but not necessarily tandem repeats one simply needs to find, for each c, the pair (A_j^r, A_ℓ^t) $(0 \le j \le r \le c$, and $c \le \ell \le t \le n)$ for which $score(A_j^r, A_\ell^t)$ is maximal, or more generally, find all locally optimal such pairs. Tandem repeats in this comparison have midpoint c and are of the form (A_j^c, A_c^t); they correspond to paths that must start anywhere in column 0 of G_c and must end anywhere in the last row (row c) of G_c. To compute the score of the highest scoring tandem repeat, the dynamic programming would be adapted as follows. $T_c[r,t]$ (defined for $r \le c$ and $t \ge c$) would hold the maximum score in $\{score(A_j^r, A_c^t) \mid j \in [0,r]\}$, and $S_c[r,t]$ would hold the index j (as a representative for the pair (j,c)), maximizing the score. The score of the best repeat is $\max_t T_c[c,t]$. Adapting Definition 2.1 to the problem of tandem repeats, we would seek all pairs (A_j^c, A_c^t) for which $score(A_j^c, A_c^t)$ is both $\max_{j' \le c} score(A_{j'}^c, A_c^t)$ and $\max_{t' \ge c} score(A_j^c, A_c^{t'})$. An equivalent definition is given below.

DEFINITION 2.2. *A tandem repeat (A_j^c, A_c^t) is locally optimal among all such repeats if the best match for A_j^c is A_c^t and vice versa. Formally, $score(A_j^c, A_c^t) = \max_{t' \ge c} score(A_j^c, A_c^{t'})$ and $score(A_j^c, A_c^t) = \max_{j' \le c} score(A_{j'}^c, A_c^t)$.*

As in the general case, locally optimal tandem repeats can be easily identified in two passes of dynamic programming. We first compute $T_c[r,t]$ and $S_c[r,t]$, as just defined. We then compute $T_c^b[j,\ell]$ (defined for $j \le c$ and $\ell \ge c$), which holds the maximum score in $\{score(A_j^c, A_\ell^t) \mid t \in [\ell, n]\}$, and $E_c[j,\ell]$, which holds the index t (representing the pair (c,t)) maximizing the score. Here the score of the best repeat is given in $\max_j T^b[j,c]$. A *tandem repeat* is locally optimal if $S_c[c,t] = j$ and $E_c[j,c] = t$.

The straightforward algorithm outlined above would identify all locally optimal tandem repeats and all locally optimal nonoverlapping repeats in $O(n^3)$ time. Our algorithm will report the same set of repeats in $O(n^2 \log n)$ time. We will treat tandem and nontandem repeats separately. Our main algorithm is tailored to tandem repeats. We then show how to modify the algorithm (without a penalty in either time or space) to report nontandem repeats as well.

A key component of our scheme is a data structure for implicitly computing in $O(mn \log m)$ time all $O(m^2 n)$ best paths from vertices $(i,0)$ (for $i \in [0,m]$) to *all other reachable vertices* in an (m,n) grid graph. In the same time bound we will also compute the weight of the $O(mn)$ best paths from vertices $(i,0)$ (for $i \in [0,m]$) to the jth row in the grid graph (for $j \ge i$). This feature appears to be essential for a fast algorithm that identifies all repeats above a given score in a string A. The methods exhibited in this article can be viewed as another example of speeding up dynamic programming by keeping and computing only a select subset of important values, as shown on several instances in the seminal articles [EG-92] and [EGa-92].

3. Computing the all source highest scoring paths in a grid graph.

3.1. Preliminary definitions. Suppose we wish to compute the highest scoring path from each of the vertices $(i,0), i \in [0,m]$, to all other vertices, or just even to all vertices on the last row of the grid graph. The latter problem has been studied in [AALM-90, AP-88] as a tool to design a parallel algorithm for computing the highest scoring path from $(0,0)$ to (m,n). They named the $O(mn)$ matrix whose $[i,j]$ entry holds the highest scoring path from node $(i,0)$ to node (m,j) in the grid graph the $DIST$ matrix. ([AALM-90] actually used this notion to denote the $(n+m)(n+m)$ highest scoring paths from the left and upper boundaries of the grid to the bottom

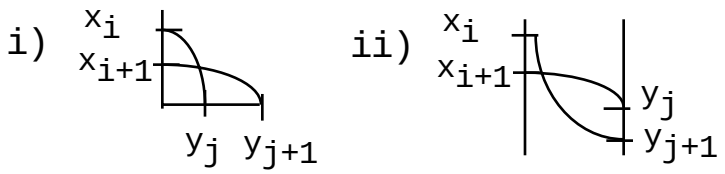


FIG. 3. *Paths that must cross.*

and right boundaries of the grid.) All our results apply to that definition as well, but for notational convenience it will be simpler to consider the smaller $DIST$ matrices.

Although [AALM-90] introduced $DIST$ matrices in the context of parallel algorithms, it turns out that $DIST$ matrices are interesting in sequential computations as well. Indeed, they were used in both [KM-93] and [B-94].

Instead of just computing one $DIST$ matrix, we will define and compute $m+1$ $DIST$ matrices associated with an (m,n) grid graph. These matrices will be denoted by $DIST_0, \ldots, DIST_m$ and are formally defined below.

DEFINITION 3.1. *Let* $DIST_\ell = [0 \ldots \ell, 0 \ldots n]$ *denote the* $(\ell+1) \times (n+1)$ *matrix, with* $DIST_\ell[i,j]$ *storing the weight of the highest scoring path from vertex* $(i,0)$ *to vertex* (ℓ, j).

None of the methods outlined in the previous sections allow us to compute all the entries in a $DIST$ matrix efficiently. Note, however, that the highest scoring path from column 0 to vertex (j,ℓ) in the grid graph corresponds to the maximum value in column j of the matrix $DIST_\ell$. Using the methods outlined in the previous section it is easy to see that the n column maxima of $DIST_\ell$ can be computed in $O(\ell n)$ time (without explicitly computing the entries of the $DIST$ matrix). Similarly using (backwards) dynamic programming, the ℓ row maxima can be computed in $O(\ell n)$ time. The naive computation of all the row and column maxima of the $m+1$ $DIST$ matrices would require $O(m^2 n)$ time. If, however, the entries of the $DIST$ matrices had been somehow precomputed their row and column maxima can be computed quickly. Indeed, Aggarwal and Park [AP-88] observed that $DIST$ matrices are Monge arrays (defined below), and [AKMSW-87] had shown that if all entries of a Monge array M can be accessed in $O(1)$ time, the m row maxima and n column maxima of M can be computed in $O(n)$ time (assuming $m \leq n$).

Following their terminology, we have the following definition.

DEFINITION 3.2. *A matrix* $M[0 \ldots m, 0 \ldots n]$ *is Monge if either condition* 1 *or* 2 *below holds for all* i, j*:*

1. $M[i,j] + M[i+1,j+1] \geq M[i,j+1] + M[i+1,j]$;
2. $M[i,j] + M[i+1,j+1] \leq M[i,j+1] + M[i+1,j]$.

The fact that $DIST$ matrices are Monge arrays (obeying property 2) is easily seen by noticing that the highest scoring paths from $x_i = (i,0)$ to $y_j = (\ell, j)$ and from $x_{i+1} = (i+1, 0)$ to $y_{j+1} = (\ell, j+1)$ (whose weights are recorded in $DIST_\ell[i,j]$ and $DIST_\ell[i+1, j+1]$) must cross (see Figure 3). These two paths can be piecewise combined to obtain a path from x_i to y_{j+1} and a path from x_{i+1} to y_j. The sum of the weights of the two highest scoring paths from x_i to y_{j+1} and x_{i+1} to y_j is by definition given by $DIST_\ell[i, j+1] + DIST_\ell[i+1, j]$. It immediately follows that $DIST_\ell[i,j] + DIST_\ell[i+1, j+1] \leq DIST_\ell[i, j+1] + DIST_\ell[i+1, j]$. For similar reasons (see Figure 3) we also have that $DIST_\ell[i+1, j] + DIST_{\ell+1}[i,j] \leq DIST_\ell[i,j] + DIST_{\ell+1}[i+1, j]$ (and hence the arrays $X_j[\ell, i] = DIST_\ell[i,j]$ are also Monge arrays, this time obeying property 1) (see Figure 3).

[AP-88] and [AALM-90] developed a recursive parallel algorithm to efficiently compute all the values in the matrix $DIST_m$. Its sequential counterpart takes $O(mn \log m)$ time and is also the best known sequential algorithm for computing $DIST_m$. We need to compute the values in $m+1$ $DIST$ matrices. If all entries of these matrices were to be output, $O(m^2 n)$ would be an obvious lower bound for the problem. This, however, turns out not to be necessary in our applications. Instead we will build a data structure in $O(mn \log m)$ time that will allow us to answer a query about the weight of the highest scoring path between $(i, 0)$ and (ℓ, j) in $O(\log \ell)$ time (for any $i \leq \ell$ and $j \in [0, n]$). Reporting *all entries* of a given $DIST$ matrix will take only $O(1)$ time per entry. The data structure will also allow us to report the n column maxima of matrix $DIST_\ell$ in $O(n + \ell \log n)$ time and the ℓ row maxima of matrix $DIST_\ell$ in $O(\ell + n \log \ell)$ time. If $m \leq n$, the total time to report all row and column maxima of the $m+1$ $DIST$ matrices is hence dominated by the time to construct the data structure and is achieved in $O(mn \log m)$ time. (For $m > n$ the time is $O(m^2 \log n)$.)

Our computation of the matrices $DIST_\ell$ proceeds as follows. Iteratively, for all values of $\ell \in [0, m]$, a representation for the matrix $DIST_\ell$ is constructed. Each such representation is computed in $O(n \log \ell)$ time from the representation of the previous $DIST$ matrix, $DIST_{\ell-1}$. Given the representation of $DIST_{\ell-1}$ the additional space required to construct the representation of $DIST_\ell$ is $O(n \log \ell)$. Hence if all representations are to be kept the total space used would be $O(mn \log m)$. In many applications, however (for example, if only the row maxima or the column maxima of the matrices $DIST_\ell$ are of interest), the representation of $DIST_{\ell-1}$ does not have to be kept after the representation of $DIST_\ell$ has been computed. We will prove the following theorems in sections 3.2–3.4.

THEOREM 3.3. *Given an (m, n) grid graph G with arbitrarily weighted edges, a representation of the m $DIST_\ell$ matrices can be computed in $O(nm \log m)$ time and $O(nm \log m)$ space. Any entry in any one of the $DIST_\ell$ matrices can thereafter be retrieved in $O(\log \ell)$ time. Retrieving an entire column of $DIST_\ell$ takes only $O(1)$ time per column entry.*

The following theorem states that row and column maxima can be computed efficiently in the above representation.

THEOREM 3.4. *Given an $m \times n$ grid graph G with arbitrary weights, all the row maxima and column maxima of the matrices $DIST_\ell$, for $\ell \in [0, m]$, can be computed in $O(nm \log m)$ time if $m \leq n$ (and $O(m^2 \log n)$ time otherwise) and $O(mn)$ space.*

3.2. Data structure used to represent the matrices $DIST_\ell$ and proof of Theorem 3.3. The data structure used to represent matrix $DIST_\ell$ is a collection $B_\ell^0, \ldots, B_\ell^n$ of $n+1$ binary trees of height $\lceil \log(\ell + 1) \rceil$. Each tree B_ℓ^j has exactly $(\ell+1)$ leaves and its tree edges are labeled by real numbers. B_ℓ^j is the tree "belonging" to vertex (ℓ, j) and its $\ell + 1$ leaves are in 1-1 correspondence (in left to right order) with the $\ell + 1$ vertices $(i, 0)$, $i \in [0, \ell]$.

In particular, *the weight of the highest scoring path from vertex $(i, 0)$ to vertex (ℓ, j) (corresponding to $DIST_\ell(i, j)$) is given by the sum of the weights of the edges on the path from the root to leaf i in B_ℓ^j (and can hence be retrieved in $O(\log \ell)$ time).*

The tree B_ℓ^j hence encodes column j of $DIST_\ell$. Note that the second part of Theorem 3.3 is readily achieved by the above representation of the $DIST$ matrices. To complete the proof of Theorem 3.3 it remains to show that the trees B_ℓ can be efficiently computed.

The structure of all trees B_ℓ is the same. B_ℓ is a subtree of the perfectly balanced

binary tree with $2^{\lceil \log(\ell+1) \rceil}$ leaves, containing only the paths to leaves $0 \dots \ell$. The tree path from the root to node i is hence (structurally) the same in all trees $B_\ell^j, j \in [0 \dots n]$ (and remains the same in trees $B_{\ell'}$ for which $\lceil \log(\ell'+1) \rceil = \lceil \log(\ell+1) \rceil$). In addition to the labels on the tree edges, each internal node x, whose left subtree is a complete binary tree, carries a label that holds the sum of the labels of the edges from x to the rightmost child in the left subtree of x.

OBSERVATION 3.5. *The ℓ highest scoring paths from nodes $(i,0), i = 1 \dots \ell$, to (ℓ, j) (i.e., the values in the jth column of the matrix $DIST_\ell$) can be output in $O(\ell)$ time by traversing the tree B_ℓ^j. Furthermore, if $c = 2^{\lceil \log(\ell+1) \rceil - 1}$ the highest scoring path from $(c,0)$ to (ℓ, j) can be retrieved in $O(1)$ time (for each j) by accessing the label of the root of B_ℓ^j.*

Note also that after traversing (and adding weights on) a path from the root to an internal node x, the highest scoring path "to the middle child" of x can be retrieved in $O(1)$ time.

We proceed to show how to construct the trees $B_{\ell+1}^0 \dots B_{\ell+1}^n$ from the trees $B_\ell^0 \dots B_\ell^n$ in $O(n \log \ell)$ time.

DEFINITION 3.6. *Denote by $B_\ell^j(i)$ the sum of the weights from the root to leaf i in B_ℓ^j. $B_\ell^j(i)$ is hence equal to $DIST_\ell[i,j]$, the highest scoring path from vertex $(i,0)$ to vertex (ℓ, j).*

The values $B_\ell^j(i) = DIST_\ell[i,j]$ are related by the following recurrence.

OBSERVATION 3.7. *Let $w_{i,j}^{k,l}$ be the weight of the edge from point (i,j) to point (k,l) in the grid graph $((k,l) \in \{(i,j+1),(i+1,j+1),(i+1,j)\})$; then:*

$$\forall\ i, \ell, j \geq 0,\ i \leq \ell + 1,\ j \leq n - 1,$$
$$B_{\ell+1}^{j+1}(i) = \max \begin{cases} B_\ell^{j+1}(i) + w_{\ell,j+1}^{\ell+1,j+1}, \\ B_\ell^j(i) + w_{\ell,j}^{\ell+1,j+1}, \\ B_{\ell+1}^j(i) + w_{\ell+1,j}^{\ell+1,j+1}, \end{cases}$$
$$B_0^{j+1}(0) = B_0^j(0) + w_{0,j}^{0,j+1},\ B_{\ell+1}^0(i) = B_\ell^0(i) + w_{\ell,0}^{\ell+1,0},\ \text{and}\ B_0^0 = 0.$$

Given the trees B_ℓ^{j+1}, B_ℓ^j, and $B_{\ell+1}^j$, the tree $B_{\ell+1}^{j+1}$ can be constructed in $O(\log \ell)$ time by adding $O(\log \ell)$ nodes to the existing structures. To construct $B_{\ell+1}^j$ we observe that the following lemma must hold.

LEMMA 3.8. *For each $j \in [0,n]$, the interval $[0, \ell+1]$, corresponding to the source vertices (rows) of the matrix $DIST_{\ell+1}$ can be partitioned into three intervals $[0, i_1)$, $[i_1, i_2)$, and $[i_2, \ell+1]$, $0 \leq i_1 \leq i_2 \leq \ell+1$, so that a highest scoring path from $(i,0)$ to $(\ell+1, j+1)$ goes through $(\ell, j+1)$ if $i \in [0, i_1)$, goes through (ℓ, j) if $i \in [i_1, i_2)$, and goes through $(\ell+1, j)$ if $i \in [i_2, \ell+1]$. (Notice that $\ell+1$ is always contained in the third interval, as the only path from $(\ell+1, 0)$ to $(\ell+1, j+1)$ goes through $(\ell+1, j)$.) Formally, there are indexes $0 \leq i_1 \leq i_2 \leq \ell+1$ so that*

$$B_{\ell+1}^{j+1}(i) = \begin{cases} B_\ell^{j+1}(i) + w_{\ell,j+1}^{\ell+1,j+1}, & i \in [0, i_1), \\ B_\ell^j(i) + w_{\ell,j}^{\ell+1,j+1}, & i \in [i_1, i_2), \\ B_{\ell+1}^j(i) + w_{\ell+1,j}^{\ell+1,j+1}, & i \in [i_2, \ell+1]. \end{cases}$$

Proof. We first give a quick proof derived from the fact that certain highest scoring paths need not cross. We then proceed to a formal proof using the Monge property which will also indicate why the points i_1 and i_2 should be efficiently computable. Suppose that a partition as described in Lemma 3.8 was not possible; then there are two points $(0,i)$ and $(0,i')$ $(i > i')$ such that all highest scoring paths from

$(\ell+1, j+1)$ to $(0, i)$ and from $(\ell+1, j+1)$ to $(0, i')$ immediately separate and then cross. These paths can be combined to obtain a pair of paths (with same score) to $(0, i)$ and $(0, i')$ that overlap up to the crossing point and hence do not immediately separate—a contradiction. The lemma can also be proven formally using the fact that $DIST$ arrays are Monge arrays, which directly implies that the following monotonicity properties hold.

Monotonicity properties. For any fixed $j < \ell$ and for all $i < \ell$, the following inequalities hold:

1. $h\_dif_i = B_\ell^{j+1}(i) - B_\ell^j(i) \geq B_\ell^{j+1}(i+1) - B_\ell^j(i+1) = h\_dif_{i+1}$,
2. $v\_dif_i = B_\ell^j(i) - B_{\ell+1}^j(i) \geq B_\ell^j(i+1) - B_{\ell+1}^j(i+1) = v\_dif_{i+1}$,
   and (adding the inequalities)
3. $d\_dif_i = B_\ell^{j+1}(i) - B_{\ell+1}^j(i) \geq B_\ell^{j+1}(i+1) - B_{\ell+1}^j(i+1) = d\_dif_{i+1}$.

The sequences (h_dif_i), (v_dif_i), and (d_dif_i), $i = 0 \ldots \ell$, are hence nonincreasing. In addition, $d\_dif_i = h\_dif_i + v\_dif_i$.

Combining the above definitions of h_dif, v_dif, and d_dif with Observation 3.7, we obtain that

$$B_{\ell+1}^{j+1}(i) = B_\ell^{j+1}(i) + w_{\ell,j+1}^{\ell+1,j+1} \quad \text{iff} \begin{cases} h\_dif_i \geq w_{\ell,j}^{\ell+1,j+1} - w_{\ell,j+1}^{\ell+1,j+1} \text{ and} \\ d\_dif_i \geq w_{\ell+1,j}^{\ell+1,j+1} - w_{\ell,j+1}^{\ell+1,j+1}, \end{cases}$$

$$B_{\ell+1}^{j+1}(i) = B_\ell^j(i) + w_{\ell,j}^{\ell+1,j+1} \quad \text{iff} \begin{cases} h\_dif_i \leq w_{\ell,j}^{\ell+1,j+1} - w_{\ell,j+1}^{\ell+1,j+1} \text{ and} \\ v\_dif_i \geq w_{\ell+1,j}^{\ell+1,j+1} - w_{\ell,j}^{\ell+1,j+1}, \end{cases}$$

$$B_{\ell+1}^{j+1}(i) = B_{\ell+1}^j(i) + w_{\ell+1,j}^{\ell+1,j+1} \quad \text{iff} \begin{cases} v\_dif_i \leq w_{\ell+1,j}^{\ell+1,j+1} - w_{\ell,j}^{\ell+1,j+1} \text{ and} \\ d\_dif_i \leq w_{\ell+1,j}^{\ell+1,j+1} - w_{\ell,j+1}^{\ell+1,j+1}. \end{cases}$$

Since h_dif_i and d_dif_i are nonincreasing it follows that $B_{\ell+1}^{j+1}(i) = B_\ell^{j+1}(i) + w_{\ell,j+1}^{\ell+1,j+1}$ holds in the interval $[0, i_1)$ for some $i_1 \geq 0$. The proof for the other cases is analogous.

Note that the values i_1 and i_2 in the lemma are generally not unique, but i_1, for example, can be chosen either as the smallest index for which $h\_dif_i < w_{\ell,j}^{\ell+1,j+1} - w_{\ell,j+1}^{\ell+1,j+1}$ or $d\_dif_i < w_{\ell+1,j}^{\ell+1,j+1} - w_{\ell,j+1}^{\ell+1,j+1}$, denoted by i_1^{max}, or it can be chosen as the smallest index for which $h\_dif_i \leq w_{\ell,j}^{\ell+1,j+1} - w_{\ell,j+1}^{\ell+1,j+1}$ or $d\_dif_i \leq w_{\ell+1,j}^{\ell+1,j+1} - w_{\ell,j+1}^{\ell+1,j+1}$, denoted by i_1^{min}. Alternatively, i_1 can be chosen as any index in $[i_1^{min}, i_1^{max}]$.

Henceforth we will assume that $i_1 = i_1^{max}$ and $i_2 = i_2^{max}$. □

LEMMA 3.9. *Tree $B_{\ell+1}^{j+1}$ can be computed from the trees B_ℓ^{j+1}, B_ℓ^j, and $B_{\ell+1}^j$ in $O(\log \ell)$ steps.*

Proof. To compute $B_{\ell+1}^{j+1}$ from the trees B_ℓ^j, B_ℓ^{j+1}, and $B_{\ell+1}^j$ the first step will consist of finding the breakpoints i_1 and i_2 (Lemma 3.8). To do so, we first perform a search on the trees B_ℓ^{j+1} and B_ℓ^j, mimicking a binary search on the values $h\_dif_i = (B_\ell^{j+1}(i) - B_\ell^j(i)), i = 0 \ldots \ell$, to determine i' the smallest i (if any) for which $h\_dif_i < w_{\ell,j}^{\ell+1,j+1} - w_{\ell,j+1}^{\ell+1,j+1}$. Clearly $i_1 \leq i'$; in fact, if the interval $[i_1, i_2)$ is not empty then $i_1 = i'$. This is so because if i_1 were less than i' we would have $h\_dif_{i_1} \geq w_{\ell,j}^{\ell+1,j+1} - w_{\ell,j+1}^{\ell+1,j+1}$, while $d\_dif_{i_1} < w_{\ell+1,j}^{\ell+1,j+1} - w_{\ell,j+1}^{\ell+1,j+1}$. But if $[i_1, i_2)$ is not empty then $v\_dif_{i_1} \geq w_{\ell+1,j}^{\ell+1,j+1} - w_{\ell,j}^{\ell+1,j+1}$. But since $d\_dif_{i_1} = h\_dif_{i_1} + v\_dif_{i_1}$, the above three inequalities cannot all hold, and $[i_1, i_2)$ not empty implies that $i_1 = i'$. Therefore, instead of determining the threshold value for d_dif_i, we proceed with a binary search

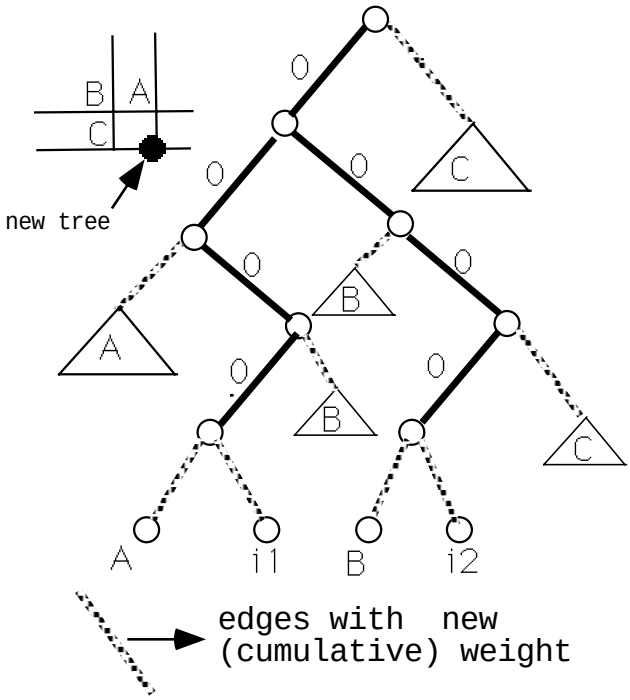


FIG. 4. *The tree $B_{\ell+1}^{j+1}$, as pasted together from subtrees of B_ℓ^{j+1} (marked A), subtrees of B_ℓ^j (marked B), and subtrees of $B_{\ell+1}^j$ (marked C) and $O(\log \ell)$ additional nodes.*

on $v\_dif_i = (B_\ell^j(i) - B_{\ell+1}^j(i)), i = 0 \dots \ell$, to find i'' the smallest i (if any) for which $v\_dif(i) < w_{\ell+1,j}^{\ell+1,j+1} - w_{\ell,j}^{\ell+1,j+1}$. For similar reasons as before, if $[i_1, i_2)$ is not empty then $i_2 = i''$. In addition, if $i'' > i'$ then $[i_1, i_2)$ cannot be empty. Hence if $i'' > i'$ we are done; otherwise the middle interval is empty and $i_1 = i_2$. The breakpoint (say i_1) between paths going through $(\ell+1, j)$ and those going through $(\ell, j+1)$ is determined by an additional binary search on $d\_dif(i) = (B_\ell^{j+1}(i) - B_{\ell+1}^j(i)), i = 0 \dots \ell$, to determine the smallest $i \in [0, \ell]$ for which $d\_dif_i < w_{\ell+1,j}^{\ell+1,j+1} - w_{\ell,j+1}^{\ell+1,j+1}$. If no such i exists, $i1 = i_2 = \ell + 1$.

After the points i_1 and i_2 have been determined, we have to construct the new tree $B_{\ell+1}^{j+1}$. (Generally, the structure of the paths to leafs $0 \dots \ell$ is the same in trees $B_{\ell+1}^{j+1}$ and B_ℓ^{j+1}. The only exception occurs when B_ℓ^{j+1} is a complete binary tree, in which case the paths in $B_{\ell+1}^{j+1}$ have an additional first edge.) $B_{\ell+1}^{j+1}$ should "contain" the portion of the tree B_ℓ^{j+1} to leaves in $[0, i_1)$ with additional weight $w_{\ell,j+1}^{\ell+1,j+1}$ and should "contain" the portion of the tree B_ℓ^j to leaves in $[i_1, i_2)$ with additional weight $w_{\ell,j+1}^{\ell+1,j+1}$, as well as the portion of the tree $B_{\ell+1}^j$ to leaves in $[i_2, \ell+1]$ with additional weight $w_{\ell+1,j}^{\ell+1,j+1}$. To construct the tree $B_{\ell+1}^{j+1}$ we start by creating a new root and new nodes along the path leading to the parent of leaf i_1 as well as the path leading to the parent of i_2. All edges on these new paths will carry a label with value 0 (see Figure 4).

Denote the path to the parent of i_1 in $B_{\ell+1}^{j+1}$ by P_1 and the path to the parent of i_2 by P_2. To complete the tree $B_{\ell+1}^{j+1}$ the appropriate subtrees from B_ℓ^{j+1}, B_ℓ^j, and

$B^j_{\ell+1}$ are attached to $P1$ and $P2$, and the edges to these subtrees must be labeled with the proper weights. Each node on $B^{j+1}_{\ell+1}$ is in a natural 1-1 correspondence with a node on $B^j_{\ell+1}$. Let L be the least common ancestor of leaf $\ell+1$ and leaf ℓ in $B^j_{\ell+1}$. With the exception of the nodes on the path from L to leaf $\ell+1$ all nodes also have a corresponding node on B^{j+1}_ℓ and B^j_ℓ. Let x be a node (internal or leaf) in B^{j+1}_ℓ, and let $B^{j+1}_\ell(x)$ denote the sum of weights on the path from the root to node x. The missing *left* subtrees of $P1$ are taken from B^{j+1}_ℓ. The label of the edge from $p \in P1$ to such a subtree rooted at x is set to $B^{j+1}_\ell(x) + w((\ell, j+1), (\ell+1, j+1))$. The missing right subtrees of $P2$ are taken from $B^j_{\ell+1}$. The label from $p \in P2$ to such a right subtree rooted at x is set to $B^j_{\ell+1}(x) + w((\ell+1, j), (\ell+1, j+1))$. All other subtrees are taken from B^j_ℓ, and the label of these edges is set to $B^j_\ell(x) + w((\ell, j), (\ell+1, j+1))$. Notice that $B^{j+1}_\ell(x)$ is only computed for children of nodes lying on the path to leaf i_1, and these values can therefore be computed in $O(\log \ell)$ time. A similar claim holds for the values $B^j_\ell(x)$ and $B^j_{\ell+1}(x)$ needed in the construction of $B^{j+1}_{\ell+1}$. In addition, each node on P_1 and P_2 that has a complete binary tree as its left subtree must compute the sum of the labels of the path to the rightmost child of this left subtree. Depending on which of the three intervals this child belongs to, this vertex label is computed (in $O(1)$ amortized time) from the labels on the new tree edges, and the vertex labels of the corresponding node in trees B^{j+1}_ℓ, B^j_ℓ, or $B^j_{\ell+1}$. The tree $B^{j+1}_{\ell+1}$ is hence constructed in $O(\log \ell)$ time and the weights on its edges and vertices satisfy the required properties. The construction of the tree $B^0_{\ell+1}$ from B^0_ℓ is slightly different (and in fact simpler) and is readily seen to be achievable in $O(\log \ell)$ time. The new nodes created are those on the path to leaf $\ell+1$. The trees B^j_0 have one leaf each and B^{j+1}_0 is constructed in $O(1)$ time from B^j_0. □

We are now ready to summarize the proof of Theorem 3.3.

THEOREM 3.3. *Given an (m, n) grid graph G with arbitrarily weighted edges, a representation of the m $DIST_\ell$ matrices can be computed in $O(nm \log m)$ time and $O(nm \log m)$ space. Any entry in any one of the $DIST_\ell$ matrices can thereafter be retrieved in $O(\log \ell)$ time. Retrieving an entire column of $DIST_\ell$ takes only $O(1)$ time per column entry.*

Proof. The representation of $DIST_\ell$ is the collection of subtrees $B^0_\ell \ldots B^n_\ell$. $DIST_\ell(i, j)$ corresponds to the weight of the path to leaf $B^j_\ell(i)$ in tree B^j_ℓ and can hence be retrieved in $O(\log \ell)$ time. By Lemma 3.9 this collection can be computed from the collection $B^0_{\ell-1} \ldots B^n_{\ell-1}$ in $O(n \log \ell)$ steps and $O(n \log \ell)$ space. The m $DIST$ matrices are hence computed in $O(nm \log m)$ time and space. To retrieve all entries of column c of $DIST_\ell$ it suffices to compute the weight of the paths to $B^c_\ell(0), B^c_\ell(2) \ldots B^c_\ell(\ell)$, which is readily achieved in $O(\ell)$ time by traversing the tree B^c_ℓ. □

3.3. Reconstructing the highest scoring paths. Given the properties established in section 3.2 it is easy to augment the grid graph with two indices per grid point enabling one to produce a (reverse) highest scoring path between a given vertex and any vertex in column 0 in time proportional to the number of vertices on the path. Each vertex in the grid graph simply records the two threshold values i_1 and i_2. To reconstruct a highest scoring path from endpoint $(\ell+1, j+1)$ to a starting point $(i, 0)$ each vertex on the path (starting with $(\ell+1, j+1)$) in turn compares i to its threshold values and determines the vertex preceding $(\ell+1, j+1)$ on the highest scoring path as follows. Whenever $i \geq i_2$ the previous vertex is $(\ell+1, j)$, whenever

$i_1 \le i < i_2$ the previous vertex is (ℓ, j) and whenever $i < i_1$ the previous vertex is $(\ell, j+1)$, until vertex $(i, 0)$ is reached.

LEMMA 3.10. *Given the row and column maxima of the matrix $DIST_\ell$, the score and endpoint of a highest scoring path from any point on row ℓ to column 0 and from any point on column 0 to row ℓ can be reported in $O(1)$ time. By augmenting each gridpoint with the two threshold values, these paths can be output in time linear in the length of the path.*

Proof. If the maximum score in row r of $DIST_\ell$ is obtained at $DIST_\ell[r, j]$, then the highest scoring path from point $(r, 0)$ to row ℓ in the grid has value $DIST_\ell[r, j]$ and ends at (ℓ, j). If the maximum score in column r of $DIST_\ell$ is obtained at $DIST_\ell[j, r]$ then the highest scoring path from point (ℓ, r) to column 0 in the grid has value $DIST_\ell[j, r]$ and ends at $(j, 0)$. The lemma follows. □

We proceed to show in the next section how to compute row and column maxima efficiently.

3.4. Row maxima in the *DIST* matrices and proof of Theorem 3.4. In order to compute the row (or column) maxima of the $DIST$ matrices we could use the algorithm described in [AKMSW-87]. They showed that if all entries of a Monge array M can be accessed in $O(1)$ time the m row maxima of M can be computed in $O(n)$ time (if $m < n$). In our model each entry is accessible in $O(\log m)$ time, which immediately yields an $O(n \log m)$ algorithm for computing all the row (as well as column) maxima. It turns out that in our model a much simpler and (by a constant factor) faster algorithm based on a simple binary search that takes advantage of the fact that some entries can be accessed in $O(1)$ time, also takes $O(n \log m)$ time. To compute the row maxima j_c of row $c = 2^{\lceil \log \ell + 1 \rceil - 1}$ in $O(n)$ time, we find the minimum of the $n+1$ values $B^\ell_j(c), j = 0 \ldots n$. Note that each $B^\ell_j(c)$ can be accessed in $O(1)$ time, since c is the rightmost child of the left subtree of the root. The row maxima of rows $i > c$ must lie in columns $j_i \le j_c$ while those in rows below c will lie in columns $j_i \ge j_c$. The row maxima for rows $i > c$ can be found in the right subtrees of the root of the trees $B^j_\ell, j = 1 \ldots j_c$, while the row maxima for rows $i < c$ can be found in the left subtrees of the trees $B^j_\ell, j = j_c \ldots n$. Furthermore, after traversing (and adding weights on) a path from the root to a node x, if x is at height r in the tree, the highest scoring path to the 2^{r-1}st child of x (generally "the middle child" of x) can be retrieved in $O(1)$ time. The m row maxima can therefore be computed in $O(n \log m)$ time (or $O(m + n \log m)$ if $m > n$).

The column maxima are computed in a similar (even simpler) fashion, in $O(m \log n + n)$ time.

We can now summarize the proof of Theorem 3.4 and its corollary.

THEOREM 3.4. *Given an $m \times n$ grid graph G with arbitrary weights, all the row maxima and column maxima of the matrices $DIST_\ell$ for $\ell \in [0, m]$ can be computed in $O(nm \log m)$ time if $m \le n$ (and $O(m^2 \log n)$ time otherwise).*

Proof. Using the algorithm of [AKMSW-87] this follows immediately from the fact that entries in $DIST_\ell$ can be accessed in $O(\log m)$ time. We have also shown that a simpler algorithm achieves the same complexity. Since the representation of $DIST_{\ell-1}$ need not be kept after the representation of $DIST_\ell$ is completed, the space requirement is only $O(mn)$.

3.5. Formal presentation of algorithm. Figure 5 gives high level pseudocode for the computation of the trees B^k_ℓ (encoding high scoring paths) and the computation of the table $Col\_to\_row\_max$, whose $[j, \ell]$ entry stores the highest scoring path from point j on the origin column to row ℓ.

```
Procedure Compute_high_scoring_paths(M[r_0 ... m, k_0 ... n])
1.   for ℓ ← r_0 to m
2.       {for k ← k_0 to n
3.            { Compute tree B_ℓ^k
4.              /* B_ℓ^k encodes the paths from points on column k_0 to (k, l) */
5.            }
6.        Compute all row maxima in DIST_ℓ    /* these correspond to best paths */
7.                                  /* to row ℓ from all points (j, k_0), j ← r_0 ... ℓ */
8.        and record in (Col_to_row_max[j, ℓ].score, Col_to_row_max[j, ℓ].index)
9.        for j ← r_0 to ℓ
10.           { Output "the best path from point (j, k_0) to row ℓ has a score of
11.             Col_to_row_max[j, ℓ].score and ends at Col_to_row_max[j, ℓ].index."
12.           }
13.      }
```

FIG. 5. *Computing the highest scoring paths from column k_0 to all other points in $M[r_0 \cdots m, k_0 \cdots m]$.*

4. Identifying tandem repeats. We now show how the above construction can be used to identify approximate repeats in a string A. We would like to report all approximate repeats that are "locally optimal" as defined in Definition 2.2 in section 2.

Recall that a tandem repeat (A_j^i, A_i^t) is locally optimal if the best match for A_j^i is A_i^t and vice versa. Alternatively, (A_j^i, A_i^t) is locally optimal if a best path from point (j, i) to row i is to (i, t) and a best path from column i to (i, t) starts from point (j, i). It immediately follows from the above definition that there are no more than $n(n-1)$ approximate tandem repeats xy that are locally optimal. (Each substring s of A can participate in at most two locally optimal repeats, one of the form sy, the other of the form xs.)

Our algorithm will report the set of locally optimal approximate repeats with weight above some predetermined threshold T.

All locally optimal approximate tandem repeats with midpoint c correspond to locally optimal paths in the grid graph that start at column c and end at row c (see Figure 6).

To identify the repeats we use an algorithm that can be viewed as an adaptation of the algorithm of [ML-84] to the approximate repeat problem together with the methods developed in section 3. Observe that all paths in the grid graph that correspond to approximate repeats either (a) cross column $n/2$ or (b) cross row $n/2$ or (c) lie entirely in the upper triangular grid graph delimited by the three points $(0, 0)$, $(0, n/2)$, $(n/2, n/2)$ or in the triangular grid graph delimited by the three points $(n/2, n/2)$, $(n/2, n)$, and (n, n) (see Figure 6).

We first find approximate repeats of types (a) and (b) and then recursively apply the algorithm to the two subcases corresponding to (c).

To find the approximate repeats that cross column $n/2$ we use the construction of section 3 twice. First, for each k, we find the highest scoring paths from nodes $(k, n/2)$ $(k = 1 \ldots c)$ to row c, by constructing the trees $B1_c^j$ $(j = n/2 \ldots n)$ and computing the c row maxima. We compute and record the best path from each point $(k, n/2)$ to row c in the table $Col\_to\_row\_max[k, c]$, as a pair (s, i), where s is the score of the highest scoring path to row c and i is the largest index for which the path from $(k, n/2)$ to (c, i) has score s. (This corresponds to computing the maximum in row

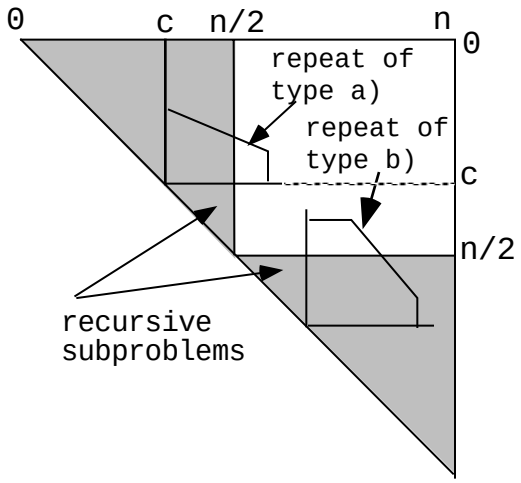


FIG. 6. *The four "types" of repeats examined by the algorithm.*

k of $DIST_c$, using a tie breaking rule.) We then use the construction (with minor modifications) to construct the highest scoring paths (in the grid graph with reversed edges) from nodes $(k, n/2)$ to (the relevant parts of) columns $n/2 \ldots 0$ by constructing the trees $B2_c^j$ $(j = 0 \ldots c)$. $B2_c^j(k)$ is the weight of the highest scoring path from (j, c) to $(k, n/2)$ and is computed for $k \in [j \cdots c]$. We compute and record the best path from each point $(k, n/2)$ to column c in $Col\_to\_col\_max[k, c]$.

Any approximate repeat whose path crosses column $n/2$ at $(k, n/2)$ is composed of two pieces: a highest scoring path from $(k, n/2)$ to row c, and a highest scoring path (in the grid graph with reversed edges) from $(k, n/2)$ to column c. We have computed and stored all these $O(n^2)$ values in $Col\_to\_col\_max[k, c]$ and $Col\_to\_row\_max[k, c]$ (for $k = 1 \ldots n/2$ and $c = k \ldots n/2$) in $O(n^2 \log n)$ time. Combining the two paths we obtain the best approximate repeat from column c to row c that crosses column $n/2$ at row k. We then repeat the above procedure to find "best tandem repeats" whose alignment path crosses row $n/2$. Finally we recursively find the repeats that lie entirely in the upper triangular grid graphs of $M[1 \cdots n/2, 1 \cdots n/2]$ and of $M[n/2 \cdots n, n/2 \cdots n]$. It is easy to verify that a repeat that is locally optimal will always correspond to such a "best repeat." Unfortunately, not all such "best repeats" will correspond to locally optimal repeats. We address this problem later. Note that the above algorithm will identify at most $O(n^2)$ candidate repeats that cross column $n/2$ (and $O(n^2)$ repeats that cross row $n/2$). The time taken to identify these candidates is $O(n^2 \log n)$. The algorithm then proceeds recursively as outlined in Figure 6 to find repeats that are contained in the two triangular grid graphs of size $n/2$. This yields an algorithm of complexity $T(n) = O(n^2 \log n) + 2T(n/2)$, which gives $T(n) = O(n^2 \log n)$. The number of candidate repeats identified is at most $C(n) = O(n^2) + 2C(n/2)$, which is $O(n^2)$.

Figure 7 summarizes how the above-described candidate set is constructed. The "highest scoring tandem repeat" corresponds to the element with maximum score in this set. A slightly modified *Compute_High_Scoring_paths* procedure (from Figure 5) is used in Figure 7 for each of the following four computations. To compute the highest scoring paths: (a) from points on the origin column c_0 to rows $r_0 \ldots r_0 + t$ we will refer to *Compute_High_Scoring_paths_col_to_row* and store the best path from (p, c_0) to row

r in $Col\_to\_row\_max[p, r]$; (b) for the highest scoring paths (in the reverse graph) from points on the origin column c_0 to columns $c_0 \ldots c_0 - t$ we will refer to *Compute_High_Scoring_paths_col_to_col* and store the best path from (p, c_0) to column c in $Col\_to\_col\_max[p, c]$; (c) for the highest scoring paths from points on the origin row r_0 to rows $r_0 \ldots r_0 + t$ we will refer to *Compute_High_Scoring_paths_row_to_row* and store the best path from (r_0, k) to row r in $Row\_to\_row\_max[k, r]$; finally, to compute the highest scoring paths from points on the origin row r_0 to columns $c_0 + t \ldots c_0$ we will call *Compute_High_Scoring_paths_row_to_col* and store the best path from (r_0, k) to column c in $Row\_to\_col\_max[k, c]$. Note that the procedure *Compute_High_Scoring_paths* in Figure 5 corresponds to *Compute_High_Scoring_paths_col_to_row*.

```
Procedure Construct_Candidates(r0, n0)
1.  k0 ← (n0 + r0) div 2
2.  Compute_High_Scoring_paths_col_to_row(M[r0 ... k0, k0 ... n0])
3.  Compute_High_Scoring_paths_col_to_col(M[r0 ... k0, r0 ... k0])
4.  for p ← r0 to k0                    /* find repeats crossing (p, k0) */
5.      for r ← p to k0
6.          {(score1, indxc) ← Col_to_row_max[p, r]
7.           (score2, indxr) ← Col_to_col_max[p, r]
8.           s ← score1 + score2        /* s is score of best repeat with midpoint r
                                           going through (p, k0) */
9.           Add ⟨(indxr, r), (r, indxc), s⟩ to Candidate_set
10.              /* ⟨(indxr, r), (r, indxc), s⟩ represents the path from (indxr, r)
                                        to (r, indxc) with score s */
11.         }
12. Compute_High_Scoring_paths_row_to_row(M[k0 ... n0, k0 ... n0])
13. Compute_High_Scoring_paths_row_to_col(M[r0 ... k0, k0 ... n0])
14. for p ← k0 to n0                    /* find repeats crossing row k0 */
15.     for r ← k0 to n0
16.         {(score1, indxr) ← Row_to_row_max[p, r]
17.          (score2, indxc) ← Row_to_col_max[p, r]
18.          s ← score1 + score2        /* s is score of best repeat with midpoint r
                                           going through (k0, p) */
19.          Add ⟨(indxr, r), (r, indxc), s⟩ to Candidate_set
20.              /* ⟨(indxr, r), (r, indxc), s⟩ represents the path from (indxr, r)
                                        to (r, indxc) with score s */
21.         }
22. Construct_Candidates(r0, k0)
23. Construct_Candidates(k0, n0)
24. ⟨(indxr, c), (c, indxc), s⟩ ← element with maximum score in Candidate_set
25. Output "a best repeat is given by path from (indxr, c) to (c, indxc) with score s";
```

FIG. 7. *Computing the highest scoring repeat.*

Although the candidate set certainly contains the set of all locally optimal repeats above the threshold score, it may also contain approximate repeats that are not locally optimal. The simplest such case occurs when two repeats x, y and x, y' (or x', y) are reported, in which case only one can be locally optimal. A trickier case occurs when the repeat x, y was reported, and there is a repeat x', y (x is a suffix of x') with higher

score which was, however, not reported because a repeat x', y' was detected with even higher score. By simply examining the candidate repeats it is not possible to detect whether x, y is indeed locally optimal. We proceed to show how to determine whether a given candidate repeat is locally optimal.

4.1. Detecting locally optimal repeats. All repeats in this section will refer to repeats with the same midpoint c, i.e., repeats corresponding to highest scoring paths from column c to row c. As discussed in section 2, to show that a tandem repeat corresponding to a highest scoring path from (i, c) to (c, j) is locally optimal, it suffices to show that the highest scoring path from (i, c) to row c ends at (c, j) and that the highest scoring path from (c, j) to column c (in the transposed graph) ends at (i, c). Suppose that the path from (i, c) to (c, j) was reported in the previous algorithm and that this path crosses column $n/2$ at row k. This path has the property that it is the highest scoring path from (i, c) to row c *through point* $(k, n/2)$ and the highest scoring path from (c, j) column c *through* $(k, n/2)$. The above path corresponds to a locally optimal repeat if and only if the highest scoring path from (i, c) to row c indeed goes through $(k, n/2)$ and so does the highest scoring path from (c, j) to column c.

Consider the following highest scoring paths matrices $R1_c^{n/2}$ and $R2_c^{n/2}$ for repeats with midpoint c that cross column $n/2$. $R1_c^{n/2}[k, j]$ is the weight of the repeat corresponding to the highest scoring path from column c through point $(k, n/2)$ to (c, j), while $R2_c^{n/2}[k, i]$ is the weight of the highest scoring path from (i, c) through $(k, n/2)$ to row c. "The best repeat" through column $n/2$ ending at (c, j) corresponds to the maximum value in column j of the matrix $R1_c^{n/2}$, while the best repeat through column $n/2$ starting at (i, c) corresponds to the maximum value in column i of $R2_c^{n/2}$. Given $Col\_to\_col\_max[c, k]$ and the trees $B1_c$ (resp., $Col\_to\_row\_max[c, k]$ and $B2_c$) we can compute a given value in the matrix $R1_c^{n/2}$ (resp., $R2_c^{n/2}$) in $O(\log n)$ time, by observing that $R1_c^{n/2}[k, j] = Col\_to\_col\_max[c, k] + B1_c^j(k)$ and $R2_c^{n/2}[k, i] = Col\_to\_row\_max[c, k] + B2_c^i(k)$. $B1_c^j(k)$ (resp., $B2_c^i(k)$) is computable in $O(\log n)$ time by traversing the path from the root to leaf k in tree $B1_c^j$ (resp., $B2_c^i(k)$).

Note that $R1_c^{n/2}$ and $R2_c^{n/2}$ have the same properties as a $DIST$ array and are therefore Monge arrays. To compute all the column maxima of the matrices $R1_c^{n/2}$ and $R2_c^{n/2}$ we could use the algorithm given in [AKMSW-87]. Alternatively, we can again also use a simpler algorithm, by taking advantage of the fact that if accessed in the right order, entries of $R1_c^{n/2}$ and $R2_c^{n/2}$ can be accessed in $O(1)$ time. Let $j_0 = 3n/4$ be the index of the middle element in the relevant portion of row c. We can find the maximum (over k) of $Col\_to\_col\_max[c, k] + B1_c^{j_0}(k)$ in $O(n)$ time (by accessing all the values in the tree $B1_c^{j_0}$). If this maximum is obtained at point $(k_0, n/2)$ then the maxima for $j < j_0$ will be at points $(k, n/2)$, with $k \geq k_0$, and those for $j > j_0$ will be at points $(k, n/2)$, with $k \leq k_0$. This clearly gives an $O(n \log n)$ time algorithm for finding all the column maxima in $R1_c^{n/2}$, and hence the total time to compute the column maxima in all matrices $R1_c^{n/2}$, $c = 1 \ldots n/2$, is $O(n^2 \log n)$. The column maxima in the matrices $R2_c^{n/2}$ are computed in the same manner.

Notice that if the computation is to use no more than $O(n^2)$ space we would first compute the $O(n^2)$ values $Col\_to\_row\_max[c, k]$ and $Col\_to\_col\_max[c, k]$ (as well as those in $Row\_to\_row\_max[c, k]$ and $Row\_to\_max\_max[c, k]$). At this point the "best tandem repeat" crossing row $n/2$ and column $n/2$ has been determined. We then recompute the trees B_c^j to determine the column maxima of $R1$ and $R2$ to find all

locally optimal repeats. The computation then recursively continues as shown in Figure 7 (lines 22 and 23).

Throughout the computation each point (c, r) will remember the best repeat (above the threshold) ending at (c, r) found so far, by setting $S[c, r] = j$ and setting $T[c, r]$ to the weight of the path from (j, c) to (c, r). Similarly, each point (j, c) will also record the best repeat starting at (j, c) found so far, by setting $E[j, c] = r$ and setting $T[j, c]$ to the weight of the path from (j, c) to (c, r). (Each gridpoint will hence store exactly two pairs of values, corresponding to the best repeat starting at that point and the best repeat ending at that point.) A repeat corresponding to a path from (j, c) to (c, r) will be reported at the end of the algorithm if $E[j, c] = r$ and $S[c, r] = j$.

5. Locally optimal nontandem repeats. As mentioned in the introduction, [KM-93] have addressed the problem of finding the (not necessarily adjacent) nonoverlapping repeat in a string. Their algorithm takes $O(n^2 \log^2 n)$ time. Our algorithm can easily be modified to address this problem as well, and can report all locally optimal repeats as defined in section 3.

The algorithm described in section 4, modified as described below, can be used to identify such repeats.

1. In addition to computing the highest scoring path from point $(k, n/2)$ to row c (resp., column c), one also computes the highest scoring path from $(k, n/2)$ *up to* row c (resp., column c). This is easily done by simply computing the maximum of $Col\_to\_row\_max[k, c']$ (resp., $Col\_to\_col\_max[k, c']$) over all $c' \leq c$. The best nontandem repeat through $(k, n/2)$ starting at (i, c) is then the highest scoring path from (i, c) to $(k, n/2)$ followed by the highest scoring path from $(k, n/2)$ to a row $c' \leq c$.
2. In addition to repeats that cross column $n/2$ and those that cross row $n/2$, and the recursive subproblems, one also needs to examine repeats that correspond to paths that lie entirely in the subgrid delimited by points $(0, n/2)$, $(0, n)$, $(n/2, n)$, and $(n/2, n/2)$; see Figure 6. For each point (i, j) in this square subgrid we need to find the highest scoring path starting anywhere in this subgrid and ending at (i, j). Note that this corresponds to a local alignment of the first half of the string with the second half. This can hence be easily achieved by running a forward and backward dynamic programming as explained in section 2.

A nontandem repeat corresponding to a path from (i, j) to (i', j') is reported, after all iterations are completed, if $E[i, j] = (i', j')$ and $S[i', j'] = (i, j)$. Note that the number of locally optimal nontandem repeats is also bounded by $O(n^2)$.

6. Grid graphs corresponding to edit graphs with unit weights. In this section we show how to efficiently use the methods of section 3 in a restricted scoring scheme. Two types of edit graphs with unit weights have been extensively studied in the literature. The first type corresponds to the Levenshtein measure [L-66]; in this measure matching symbols score 0, while substitutions/deletions score 1. This measure has been proven to provide combinatorial leverage not found in other scoring schemes when seeking alignments with a weight below a given threshold k, for example, in [LV-88], [My-86], [LS-93], and [LMS-94]. A seemingly similar but in fact much more sensitive measure is obtained when deletions and substitutions have (*constant*) weight $-S$, for some integer S, and matching symbols have (*constant*) integer weight M (typically for very small integers S and M), and a maximum cost alignment is sought. A maximum weight alignment in this second model will favor long alignments

with k deletions/substitutions over short alignments with (say) the same number of substitutions; in particular, alignments with a score above 0 will have a fraction of $\frac{S}{M+S}$ matching symbols.

The methods developed in section 3 can be modified to compute some representation of the matrices $DIST_\ell, \ell = 1 \ldots, m$, of an (m, n) grid graph in $O(mn)$ time. In the current special case a representation of $DIST_\ell$ can be computed in $O(n)$ time, when given an appropriate representation of $DIST_{\ell-1}$. The representation will allow us to augment the grid graph as outlined in section 3.3, so that any highest scoring path can be produced in time proportional to its length. However, general queries about the score of a highest scoring path will also require $O(n)$ time and hence will require as much time to be answered as it takes to produce the path itself. We will show that this limited capability, while not yielding a faster algorithm for the approximate repeat problem, does yield a faster algorithm for the cyclic string comparison (defined in the next subsection).

It is easy to see that if all horizontal and vertical edges in a grid graph have weight $-S$, and diagonal edges have weight M or $-S$, then $DIST_\ell[i, j+1] \geq DIST_\ell[i, j] - S$ and $DIST_\ell[i, j] \geq DIST_\ell[i, j+1] - S - M$. It immediately follows that $DIST_\ell[i, j+1] - DIST_\ell[i, j]$ assumes integer values in $[-S, S+M]$. Similarly, it is not hard to see that $DIST_\ell[i, j] - DIST_{\ell+1}[i, j]$ assumes integer values in $[-S-M, S]$. Instead of actually computing the values $DIST_\ell[i, j], i = 1 \ldots \ell$, we will only compute the differences $DIST_\ell[i, j+1] - DIST_\ell[i, j], i = 1 \ldots \ell$, and $DIST_\ell[i, j] - DIST_{\ell+1}[i, j], i = 1 \ldots \ell$. Since these differences assume at most $2S + M = O(1)$ values and are nonincreasing for $i \in [0, m]$, they can be represented as a list of $O(1)$ elements $I_0, I_1, \ldots, I_P$, $P = 2S+M+1$, such that $(DIST_\ell[i, j+1] - DIST_\ell[i, j]) = -S+j$ in the interval $[I_j, I_{j+1})$.

THEOREM 6.1. *Given a grid graph G with edge weights in $\{-S, M\}$ (for integer values M and S) and in which all horizontal and vertical edges have weight S, the row maxima and column maxima of the matrix $DIST_n$ can be computed in $O(nm)$ time.*

Proof (sketch). Let i' be the smallest $i \in [0, \ell]$ for which $DIST_\ell[i, j+1] - S < DIST_\ell[i, j] + w_{\ell,j}^{\ell+1,j+1}$, or $\ell + 1$ if there is no such i. Let i'' be the smallest $i \in [0, \ell]$ for which $DIST_\ell[i, j] + w_{\ell,j}^{\ell+1,j+1} < DIST_{\ell+1}[i, j] - S$, or $\ell + 1$ if there is no such i. If $i'' \geq i'$ then let $i_1 = i'$ and $i_2 = i''$. Otherwise let i_2 be the smallest i for which $DIST_\ell[i, j+1] - DIST_{\ell+1}[i, j] < 0$, and set $i_1 = i_2$.

Given the intervals in which $DIST_\ell[i, j+1] - DIST_\ell[i, j]$ is constant and the intervals in which $DIST_\ell[i, j] - DIST_{\ell+1}[i, j]$ is constant, the values i_1 and i_2 can clearly be computed in $O(1)$ time. Given these values we can, as in the case of general weight matrices, break down the general recurrence from Observation 3.7 into the following three cases.

$$DIST_{\ell+1}[i, j+1] = \begin{cases} DIST_\ell[i, j+1] - S, & i \in [0, i_1), \\ DIST_\ell[i, j] + w_{\ell,j}^{\ell+1,j+1}, & i \in [i_1, i_2), \\ DIST_{\ell+1}[i, j] - S, & i \in [i_2, \ell+1]. \end{cases}$$

Note, however, that the above equalities cannot be used directly to actually compute $DIST_{\ell+1}[i, j+1]$, as $DIST_\ell[i, j+1]$ is not directly available; only the differences $DIST_\ell[i, j+1] - DIST_\ell[i, j]$ and $DIST_\ell[i, j] - DIST_{\ell+1}[i, j]$ are available, which was sufficient for computing the breakpoints i_1 and i_2. From the above equalities, given the differences $DIST_\ell[i, j+1] - DIST_\ell[i, j]$, $DIST_\ell[i, j] - DIST_{\ell+1}[i, j]$, the differences $DIST_\ell[i, j+1] - DIST_{\ell+1}[i, j]$ and $DIST_{\ell+1}[i, j] - DIST_{\ell+1}[i, j]$ can now be computed in $O(S + M) = O(1)$ time (for each point $(\ell + 1, j + 1)$). In addition, each node $(\ell + 1, j + 1)$ in the grid can be augmented as in section 3.3 with the threshold

values i_1 and i_2 that indicate whether the highest scoring path to $(i, 0)$ for a given i is to go through $(\ell, j+1)$, (ℓ, j), or $(\ell+1, j)$. After computing the above representation of the matrix $DIST_\ell$ from $DIST_{\ell-1}$, one could compute all values in that matrix in $O(\ell n)$ time. and at the same time compute all the row and column maxima. Alternatively, one could compute the values in any row i in $O(n)$ time. Note that $DIST[i, 0] = (m-i)S$ for any i. We can hence compute the n values in this row from the differences $DIST_m(i, j+1) - DIST_m(i, j)$ in $O(n)$ time. □

6.1. Application to the cyclic string comparison problem. Given two strings A and B the cyclic string comparison problem consists of finding the cyclic shift of A which can be best aligned with B. Formally, if $B = B_1 \dots B_n$, we wish to find the i for which the optimal alignment of $A = A_i, A_{i+1}, \dots A_m, A_1, \dots A_{i-1}$ with $B_1, \dots B_n$ gives the best score. For general weight matrices, this problem was addressed in [Ma-90] where an $O(mn \log m)$ algorithm was presented. The techniques in [AALM-90] result in a parallel algorithm for that problem. [LMS-94] have shown how to solve the cyclic string comparison problem under the Levenshtein measure in $O(n+km)$ time provided that the cost of the best alignment is bounded by k. Here we consider the problem when all values in the weight matrix are restricted to an interval $[-S, M]$ for positive constants S and M. The cyclic string comparison problem can be reduced to the following highest scoring paths problem. Consider computing the $(2m \times n)$ grid graph of the comparison of the string AA (A concatenated with A) with B. Consider the matrix $DIST_m$ of that grid graph. The weights of the best alignments of the cyclic shifts of A with B are given by the values $DIST_{m+i}[i, n], i = 0 \dots m-1$. After computing the representation of the $DIST$ matrices described in the previous section in $O(mn)$ time, each value $DIST_{m+i}[i, n]$ can be computed in $O(n)$ time. The cyclic shift of A that best matches B and its alignment with B can hence be produced in $O(mn)$ time.

7. Conclusions. The algorithms presented in this paper allow the quick computation of certain highest scoring paths in grid graphs. We have given two string matching applications in which such paths can help solve interesting sequence comparison problems, while improving upon previous algorithms by a factor of $O(\log n)$. It would be interesting to find additional applications for these methods. An interesting question that remains open is whether approximate repeats can be found quicker in the simple scoring schemes considered in the last section.

Acknowledgments. I would like to thank Dina Kravets and Gad Landau for discussions on the subject, and Raffaele Giancarlo for valuable comments on an earlier version of this manuscript.

REFERENCES

[AALM-90] A. Apostolico, M. Atallah, L. Larmore, and S. McFaddin, *Efficient parallel algorithms for string editing problems*, SIAM J. Comput., 19 (1990), pp. 968–988.

[AKMSW-87] A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilber, *Geometric applications of a matrix-searching algorithm*, Algorithmica, 2 (1987), pp. 195–208.

[AP-88] A. Aggarwal and J. Park, *Notes on searching in multidimensional monotone arrays*, in Proc. 29th IEEE Symp. on Foundations of Computer Science, 1988, pp. 497–512.

[AP-83] A. Apostolico and F. P. Preparata, *Optimal off-line detection of repetitions in a string*, Theoret. Comput. Sci., 22 (1983), pp. 297–315.

[B-94] G. Benson, *A space efficient algorithm for finding the best non-overlapping alignment score*, in Proc. Fifth Ann. Symp. on Combinatorial Pattern Matching, Lecture Notes in Computer Science 807, Springer-Verlag, New York, 1994, pp. 1–14.

[C-81] M. Crochemore, *An optimal algorithm for computing repetitions in a word*, Inform. Process. Lett., 12 (1981), pp. 244–250.

[EG-92] D. Eppstein, Z. Galil, R. Giancarlo, and G. Italiano, *Sparse dynamic programming* II: *Convex and concave cost functions*, J. Assoc. Comput. Mach., 39 (1992), pp. 546–567.

[EGa-92] D. Eppstein, Z. Galil, R. Giancarlo, and G. Italiano, *Sparse dynamic programming* I: *Linear cost functions*, J. Assoc. Comput. Mach., 39 (1992), pp. 519–545.

[ES-83] B. W. Erickson and P. H. Sellers, *Recognition of patterns in genetic sequences*, in Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison, D. Sankoff and J. B. Kruskal, eds., Addison-Wesley, Reading, MA, 1983, pp. 55–91.

[H-88] X. Huang, *A lower bound for the edit-distance problem under arbitrary cost function*, Inform. Process. Lett., 27 (1988), pp. 319–321.

[KM-93] S. K. Kannan and E. W. Myers, *An algorithm for locating non-overlapping regions of maximum alignment score*, in Proc. Fourth Ann. Symp. on Combinatorial Pattern Matching, Lecture Notes in Computer Science 684, Springer-Verlag, New York, 1993, pp. 74–86.

[L-66] V. I. Levenshtein, *Binary codes capable of correcting deletions, insertions and reversals*, Soviet Phys. Dokl., 10 (1966), pp. 707–710.

[LMS-94] G. Landau, G. Myers, and J. Schmidt, *Incremental string comparison*, SIAM J. Comput., 27 (1998), to appear.

[LS-93] G. M. Landau and J. P. Schmidt, *An algorithm for approximate tandem repeats*, in Proc. Fourth Ann. Symp. on Combinatorial Pattern Matching, Lecture Notes in Computer Science 684, Springer-Verlag, New York, 1993, pp. 120–133.

[LV-88] G. M. Landau and U. Vishkin, *Fast string matching with k differences*, J. Comput. System Sci., 37 (1988), pp. 63–78.

[M-92] W. Miller, *An Algorithm for Locating a Repeated Region*, manuscript.

[Ma-90] M. Maes, *On a cyclic string-to-string correction problem*, Inform. Process. Lett., 35 (1990), pp. 73–78.

[ML-84] M. G. Main and R. J. Lorentz, *An $O(n \log n)$ algorithm for finding all repetitions in a string*, J. Algorithms, 5 (1984), pp. 422–432.

[My-86] E. Myers, *An $O(ND)$ difference algorithm and its variations*, Algorithmica, 1 (1986), pp. 251–266.

[SW-81] T. F. Smith and M. S. Waterman, *Identification of common molecular subsequences*, J. Molecular Biol., 147 (1981), pp. 195–197.

[WE-87] M. S. Waterman and M. Eggert, *A new algorithm for best subsequence alignment with application to tRNA-rRNA comparisons*, J. Molecular Biol., 197 (1987), pp. 723–728.

SIAM J. COMPUT.
Vol. 27, No. 4, pp. 993–1015, August 1998
© 1998 Society for Industrial and Applied Mathematics
005

BOUNDING THE POWER OF PREEMPTION IN RANDOMIZED SCHEDULING*

RAN CANETTI† AND SANDY IRANI‡

Abstract. We study on-line scheduling in overloaded systems. Requests for jobs arrive one by one as time proceeds; the serving agents have limited capacity and not all requests can be served. Still, we want to serve the "best" set of requests according to some criterion. In this situation, the ability to *preempt* (i.e., abort) jobs in service in order to make room for better jobs that would otherwise be rejected has proven to be of great help in some scenarios.

We show that, surprisingly, in many other scenarios this is not the case. In a simple, generic model, we prove a polylogarithmic lower bound on the competitiveness of randomized and preemptive on-line scheduling algorithms. Our bound applies to several recently studied problems. In fact, in certain scenarios our bound is quite close to the competitiveness achieved by known *deterministic, nonpreemptive* algorithms.

Key words. randomized algorithms, scheduling, preemption, lower bounds

AMS subject classifications. 68M10, 68M20, 90B12

PII. S0097539795283292

1. Introduction.

Scheduling problems pervade many aspects of system design and management. Consider, for instance, the following problems.

(a) A system with several processors is assigned jobs of varying duration and load, where each job is to be performed on a single processor. The jobs arrive one by one and each job must be assigned to a processor before the next job is known. The goal is to assign the jobs to processors "in the best possible way."

(b) A set of gateways connects a network of computers to a set of peripheral devices. At any point in time, a node in the network may request a connection to a particular type of device for some period of time. The bandwidth required for the different connections vary. The goal is, again, to assign connections to gateways "in the best possible way."

(c) A communication network with guaranteed bandwidth policy (e.g., asynchronous transfer mode (ATM) [23]) services many different types of traffic. Requests for connections between nodes in the network arrive and depart through time. The durations, priorities, and bandwidth requirements of connections vary (e.g., e-mail, video, etc.). Connections should be allocated bandwidth and routed "in the best possible way."

A prominent characteristic of all these scenarios is that the scheduling algorithm learns about the incoming requests as they enter the system one by one, and must decide how to handle each task without knowledge of future tasks. Any such algorithm which must make decisions without complete knowledge of the entire input sequence is said to be an *online* algorithm.

*Received by the editors March 20, 1995; accepted for publication (in revised form) May 2, 1996; published electronically May 18, 1998.

http://www.siam.org/journals/sicomp/27-4/28329.html

†T. J. Watson Research Center, Yorktown Heights, New York (canetti@watson.ibm.com). This author was supported by American-Israeli Binational Science Foundation grant 92-00226.

‡Department of Information and Computer Science, University of California, Irvine, CA 92717 (irani@ics.uci.edu). This author was supported in part by NSF grants CCR-9309456 and GER-94-50142.

Two natural optimization problems arise from these and other related scenarios. First, we may assume that all requests will be served and aim at minimizing the maximum load on any serving agent (e.g., processor, gateway, link) at any point in time. We call this problem *load balancing*. Alternatively, we may consider a situation where the capacity of the system is limited and not all requests can be served. Here the goal is to schedule the subset of requests with maximum value according to some criterion. We call this problem *admission control*. Both optimization problems have been studied in several models. The load balancing problem emerging from example (a) above is an old problem introduced in [14] and extensively studied since [1, 6, 9, 16]. The admission control problem emerging from this example is studied in [10, 11, 22]. The load balancing problem emerging from example (b) is studied in [5, 7] and from example (c) in [1]. The admission control problem emerging from example (c) has been extensively studied (e.g., [2, 3, 12, 13, 8]). In this work we address admission control problems. (We refer the reader to [10] for an exposition of the importance of admission control.)

A natural question in such scenarios has to do with the power of *preemption.* The ability to alter previous decisions in certain ways has proven to be very powerful (say, when algorithms are allowed to reassign a job to a different server at some cost [20, 19, 15]). We address the following notion of preemption, natural in limited capacity scenarios. We allow a job to be aborted in the middle of execution in order to make room for a more valuable job that would otherwise be rejected. However, no credit is accrued for uncompleted jobs. Preempting jobs in the middle of execution may be problematic in some scenarios (say, when service is guaranteed upon admission). In other scenarios, preemption seems to be acceptable and even very helpful (say, in systems that support real-time jobs, e.g., [21]).

It has been demonstrated that an appropriate use of preemption helps considerably to enhance the *throughput* in certain settings [8]. We study the following question. To what extent can the ability to preempt jobs enhance the performance in more general cases (e.g., when the criteria for performance are different, or when the setting is different)?

We provide some surprising answers to this question by demonstrating that preemption does not help much (if at all) in a large variety of on-line admission control problems. First we consider a generic model for on-line admission control. In this model we show a lower bound on the competitiveness of any *randomized, preemptive* on-line scheduling algorithm. This bound applies, via several reductions, to a large variety of models, and in particular to the admission control problems emerging from the three examples above. We elaborate on the different scenarios where our result applies in section 1.1.

We describe our generic model. A server is given a sequence of job requests, arriving one by one as time proceeds. The server can serve only one job at a time. Each job is characterized by its arrival time, its *duration* (known upon arrival), and its *value.* A job has to be either rejected or served immediately for the specified duration. The server can *preempt* (i.e., abort) jobs in service. The value gained from a sequence is the sum of the values of completed jobs. No value is gained for preempted jobs. We consider randomized scheduling algorithms (or *schedulers*). A scheduler is f-competitive if for any sequence S of jobs the value gained by the best (off-line) schedule on S, divided by the value gained by the online scheduler on S, is at most f. Note that f may be a function of the request sequence, rather than a constant. In this model, we prove the following bound.

THEOREM 1. *Any randomized, preemptive scheduler has a competitive ratio of at best*

$$\frac{1}{10}\sqrt{\frac{\log \mu}{\log \log \mu}},$$

where μ is any of the following two measures:

(i) $\mu = \mu_v$, *the ratio between the largest and smallest values of a job in the request sequence;*

(ii) $\mu = \mu_d$, *the ratio between the largest and smallest durations of a job in the request sequence.*

We actually prove a stronger fact than suggested by the theorem: for any randomized preemptive scheduler $\mathcal{A}$ and for any μ, we construct a sequence S which simultaneously satisfies $\mu_d(S) \leq \mu$ and $\mu_v(S) \leq \mu$, such that the competitiveness of $\mathcal{A}$ on S is $1/10\sqrt{\log \mu / \log \log \mu}$. An identical bound applies also to scenarios where more than one job can be served at a time. In section 5 we describe a simple, randomized, preemptive $O(\log(\min\{\mu_d, \mu_v\}))$-competitive scheduler in our generic model, thus demonstrating that our bound is at most a roughly quadratic factor from optimality.

Many previously studied scheduling problems can be reduced to this generic model. In particular, our bound applies to the setting in which Awerbuch, Azar, and Plotkin show a scheduler with competitiveness logarithmic in μ_d and μ_v [2]. Their scheduler is both *deterministic* and *nonpreemptive.* Thus, in their setting, the combined power of randomization and preemption results in at most a roughly quadratic improvement. (We note, however, that the scheduler in [2] does not apply to the setting of Theorem 1.)

Our proof of the bound is nontrivial and occupies most of this paper. It involves techniques which we believe are of independent interest. For each randomized scheduler, we construct a request sequence on which the scheduler performs poorly relative to the best strategy. We stress that the sequence is fixed for this scheduler; that is, the sequence does not change in different runs of the scheduler. The sequence is constructed one request at a time via an interaction with the scheduler, $\mathcal{A}$. Each next request is generated based on the probability distribution of the job that $\mathcal{A}$ currently has in service. More specifically, at each step we keep a set of threshold values. We generate the next request depending on whether the probability that $\mathcal{A}$ has some specific job in service is above or below one of the thresholds. Our adversarial strategy is a randomized adaptation of a simple deterministic lower bound. Our technique may prove useful in transforming similar deterministic lower bounds into randomized ones.

1.1. Applications of the bound. We describe how our lower bound applies to several recently studied admission control problems. In section 4 we formally state and prove these applications.

Consider the setting of example (a) above. A system consists of a set of processors, each with limited capacity. Jobs of varying load, duration, and value arrive through time, each with a deadline. At all times the sum of the loads of the jobs in service on a given processor must not exceed its capacity. A scheduler must decide which jobs to execute (and on which processor) in order to complete the set of jobs with the largest total value before their deadlines. Via a simple reduction, our lower bound applies to this setting, with any number of processors, where μ is either μ_d or μ_v. A lower bound of $\Omega(\sqrt{\mu_v})$ on the competitive ratio of any *deterministic* preemptive scheduler

is shown in [11, 18], where the value of a job is arbitrary. However, their bound does not apply to randomized schedulers. Our bound holds even for the cases where:

(1) a single job cannot occupy more than a predefined fraction δ of the capacity, for any $\delta > 0$ (see Corollary 1);

(2) each job has a value equal to its duration, as opposed to having an arbitrary value;

(3) all jobs have the same value.

In the last two cases, the bound holds with respect to μ_d and μ_l, where μ_l is the ratio of largest to smallest load of a job in the input sequence. (See Corollaries 2 and 3.) Surprisingly, our bound does *not* apply when the value of each job equals its load times its duration. In fact, constant competitive algorithms exist in this case [10, 11, 22, 17, 18, 8]. We suggest an explanation for this phenomenon below.

We note that example (b) given at the beginning of the introduction is a generalization of this multiprocessor scheduling problem. The bound applies, yielding similar results.

Next we address *call control* and *virtual circuit routing* problems. Here we have a communication network with guaranteed bandwidth policy in which the links have limited capacities. Requests for connections (or *calls*) arrive through time, where each call has its source and destination nodes, as well as duration, bandwidth requirement, and value. The scheduler must route accepted calls within the capacity limitations of the links; that is, the sum of the bandwidth of calls using each link must be at all times less than its capacity.

Awerbuch, Azar, and Plotkin show a deterministic, nonpreemptive $O(\log \mu)$-competitive scheduler in this model, where $\mu = \mu_d \cdot \mu_v \cdot n$ and n is the number of nodes in the network [2]. They also show that this is the best that any deterministic, nonpreemptive scheduler can achieve. Their scheduler has the drawback that every job is constrained to require bandwidth at most $1/\log \mu$ of the capacity of any edge. Awerbuch et al. remove this constraint for networks with a tree topology via a randomized, nonpreemptive scheduler [3]. Awerbuch et al. improve the bounds for trees and give randomized, nonpreemptive schedulers for meshes [4].

Garay and Gopal initiated the study of preemptive call control in [12]. They show constant competitive preemptive schedulers for simple networks and value functions. Garay et al. show competitive schedulers on a single link and a line network, for the special case that at most one call can be accommodated on any link, and for several specific ways for determining the value of a call [13]. Bar-Noy et al. generalize their results by showing constant competitive schedulers on a single link when the value of a call is the bandwidth times the duration and every call has a bandwidth requirement which is at most a limited fraction of the capacity of the link [8]. Their strategies apply also to line networks if all calls have infinite durations. (Here the bandwidth times the duration of a call reflects the "amount of information" contained in the call. Thus, preemption helps when the quantity to be maximized is the throughput of the link.)

We provide a lower bound of $1/10\sqrt{\log \mu / \log \log \mu}$ on the competitiveness of any preemptive, randomized scheduler for any network, when the value of a call is arbitrary. The bound holds even when a single call cannot occupy more than a predefined fraction δ of the link capacity (for any $\delta > 0$), and even if the value of a call is its duration or if all calls have the same value (see Corollary 4).

Furthermore, if the network has no cycles then the bound applies with $\mu = \min\{\mu_d, D\}$, where D is the diameter of the network. In this case, the bound holds

even when the value of a call equals the duration times bandwidth, or the distance from source to destination, or the distance times duration, or the distance times duration times bandwidth (see Corollary 6).

We suggest the following explanation of this "dichotomy." The bandwidth times duration measures the "amount of information" contained in a call, whereas the bandwidth times duration times distance measures the "work" invested in a call. It can thus be said that, in the single link case, when the value of a job is directly proportional to the information contained in it then the bound does not apply and constant competitive algorithms exist. If the value of a job is determined in any other way then the bound applies. In more complex networks (even in a line of links), the bound applies even when the value of a call is directly proportional to the amount of information, or to the work invested.

Organization. In section 2 we formally define our generic model. In section 3 we state and prove the basic lower bound as stated in Theorem 2. (In section 3.1 we first prove a weaker version of the bound; the proof of this weaker version is simpler and offers intuition for the proof of the full bound.) In section 4 we state and prove several corollaries of the bound. In section 5 we demonstrate the tightness of the bound by sketching a scheduler in our model, with logarithmic competitiveness.

2. The model. We formalize the generic model described in the Introduction. A *server* is given a sequence of job requests, arriving one by one as time proceeds. We assume that time is discrete, although several requests may arrive at a single time unit. The server can serve at most one job at a time. Each job c is characterized by its arrival time t_c, its duration d_c (known upon arrival), and its *value* v_c. The scheduling of jobs is subject to the following rules. A job has to be either served immediately for the specified duration or rejected. The server can *preempt* (i.e., abort) a job in service. The server accrues an additive gain of v_c for each completed job c. No gain is accrued for preempted jobs.[1]

A sequence $S = c_1, \ldots, c_n$ of job requests is *timely* if $t_{c_i} \geq t_{c_j}$ for every $i > j$. Say that S is *feasible* if no two jobs intersect in time; that is, for no $i \neq j$ we have $t_i < t_j < t_i + d_i$. The *gain* of S is $G(S) \stackrel{\triangle}{=} \sum_{c \in S} v_c$. The *optimal feasible gain* of S is $\mathcal{O}(S) = \max_{\{S' \subseteq S | S' \text{ feasible}\}} G(S')$, where $S' \subseteq S$ means that S' is a subsequence of S.

For a scheduling algorithm $\mathcal{A}$, let $\mathcal{A}(S, r) \subseteq S$ be the sequence of jobs completed by $\mathcal{A}$ on sequence S and random input r. Algorithm $\mathcal{A}$ is a valid *scheduler* if whenever S is timely, $\mathcal{A}(S, r)$ is feasible for all r.

DEFINITION 1. *Let $\mu_1(\cdot), \ldots, \mu_t(\cdot)$ be a set of measure functions from request sequences into the reals. A scheduler $\mathcal{A}$ is* f-competitive with respect to $\mu_1(\cdot), \ldots, \mu_t(\cdot)$ *if for all large enough m and for all timely sequences S with $\max_{i=1}^{t}\{\mu_i(S)\} \leq m$ we have*

$$f(m) \geq \frac{\mathcal{O}(S)}{\mathrm{E}_r(G(\mathcal{A}(S, r)))},$$

where $\mathrm{E}_r(G(\mathcal{A}(S, r)))$ denotes the expected value of $G(\mathcal{A}(S, r))$ when r is uniformly chosen from the set of random inputs of $\mathcal{A}$. We stress that the sequence S does not depend on the random choices of $\mathcal{A}$ (i.e., using standard terminology, the adversary is *oblivious*).

[1] It may be helpful to visualize each job c as a rectangle with length d_c and height v_c/d_c; the rectangle is located on the number line so that its left edge is at point t_c. The area of the rectangle is v_c. Thus the goal is to maximize the total area of completed jobs. See Figure 3.1.

We sometimes use $\mathrm{E}G\mathcal{A}(S)$ to shorthand $\mathrm{E}_r(G(\mathcal{A}(S,r)))$. We also say that $\frac{\mathcal{O}(S)}{\mathrm{E}G\mathcal{A}(S)}$ is the *competitive ratio of* $\mathcal{A}$ *on sequence* S. Below we use the following simple observation. For a sequence S, a scheduler $\mathcal{A}$, and a job $c \in S$, let I_c denote the binary random variable having value 1 iff job c is completed in a run of $\mathcal{A}$ on S. Then,

$$\mathrm{E}_r(G(\mathcal{A}(S,r))) = \mathrm{E}_r\left(\sum_{c\in S} I_c \cdot v_c\right) = \sum_{c\in S} E_r(I_c)\cdot v_c = \sum_{c\in S} v_c \cdot \mathrm{Prob}(\mathcal{A} \text{ completes } c).$$

That is, the expected gain of the scheduler is the sum over all jobs in the input sequence of the probability that the job is completed times its value.

3. The lower bound. Let $g(x) = \frac{1}{10}\cdot\sqrt{\frac{\log(x)}{\log\log(x)}}$. Let $\mu_d(S)$ (resp., $\mu_v(S)$) be the ratio between the largest and smallest durations (resp., value) of a job in the request sequence S. We prove the following bound.

THEOREM 2. *Any randomized, preemptive on-line scheduler is at best g-competitive, with respect to measures μ_d and μ_v.*

For the proof we construct, for each scheduling algorithm $\mathcal{A}$ and infinitely many values m, a timely sequence S with $\max\{\mu_d(S),\mu_v(S)\} \le m$, and we show a feasible "off-line schedule" $S' \subseteq S$ such that $\frac{G(S')}{\mathrm{E}G\mathcal{A}(S)} \ge g(m)$. We first present a very rough description of the construction. Say that two jobs are of the same *type* if they have the same duration and value. The sequence S consists of several different types of jobs. Let $p_i(t)$ denote the probability that a job of type i is being served by $\mathcal{A}$ at the end of time unit t. Let $\vec{p}(t) = \langle p_0(t),\ldots,p_k(t)\rangle$, where the number of different types of jobs is $k+1$. Since at most one job can be scheduled at a time, we have $\sum_{i=0}^{k} p_i(t) \le 1$ for all t. Given $\mathcal{A}$, $\vec{p}(t)$ is a function only of the prefix of S consisting of the jobs requested up to time t.

The construction of the sequence S can be pictured as an interactive game between the scheduler and an adversary, where in each time unit t the adversary generates some requests based on $\vec{p}(t-1)$ and the jobs requested so far. Next the scheduler generates $\vec{p}(t)$ based on the new requests and the history of the interaction, subject to the conditions that $\sum_{i=0}^{k} p_i(t) \le 1$, and $p_i(t) \le p_i(t-1)$ unless a new i-job is requested at time t. (Here we give the scheduler some extra "leeway" by letting it know in advance all the jobs requested during the entire time unit.) Sequence S is now the concatenation of the jobs requested by the adversary in the game. We stress that S is fixed for each scheduler; it does not depend on the random choices of the scheduler in a specific run.

The adversary strategy in the above game consists of several recursive applications of roughly the same scheme. In order to better present the construction and analysis, we first describe a simpler adversary that consists of only one application of this scheme. This simpler adversary, called a 1-*adversary*, shows a weaker result than Theorem 2, namely, that no valid scheduler is less than $(\frac{\mathrm{e}+1}{\mathrm{e}} - o(1))$-competitive (where e is the base of natural logarithms).

3.1. A 1-adversary. A 1-*adversary* generates, for each scheduler $\mathcal{A}$ and each value m, a sequence S with $\mu_d(S) = m^2$ and $\mu_v(S) = m$, and a feasible "off-line schedule" $S' \subseteq S$ such that $\frac{G(S')}{\mathrm{E}G\mathcal{A}(S)} \ge \frac{\mathrm{e}+1}{\mathrm{e}}\cdot\left(1 - O(\frac{1}{m})\right)$.

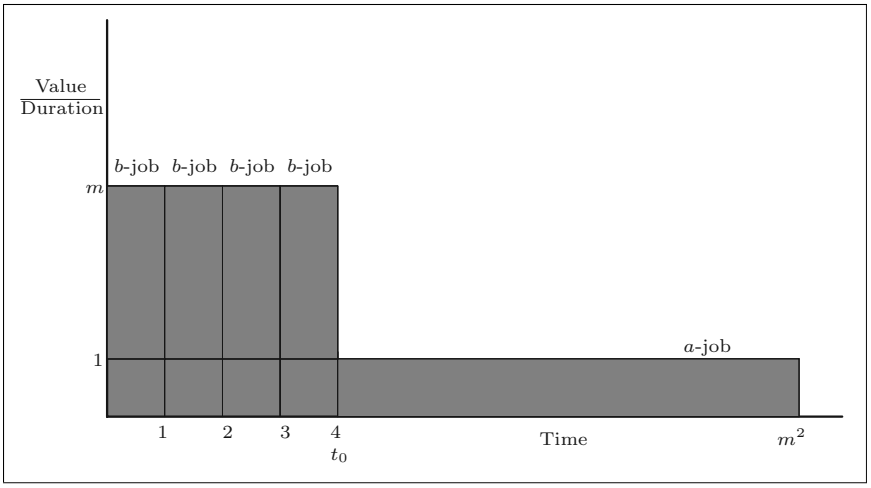


FIG. 3.1. *A* 1*-adversary. b-jobs are requested until time* t_0.

The 1-adversary uses only two types of jobs: *a-jobs* have duration m^2 and value m^2; *b-jobs* have duration 1 and value m. The sequence S is constructed as follows. The adversary first requests an a-job at time 0. Next, at each time $t = 0, \ldots, t_0$ (where t_0 is computed below) a b-job is requested. Note that any feasible subsequence of S consists of either the a-job or some of the b-jobs (see Figure 3.1).

Setting an appropriate "stopping time," t_0, is the crux of the adversarial strategy. If the scheduler were deterministic (or alternatively, if the adversary could see the random choices of the scheduler), then computing t_0 would be simple. If $\mathcal{A}$ preempts the a-job in favor of some b-job, the input sequence would stop. This way, $\mathcal{A}$ gains m while the optimal schedule is the a-job, with gain at least m^2. If $\mathcal{A}$ never preempts the a-job then we set $t_0 = m^2$ (that is, b-jobs are requested during the entire duration of the a-job), and the optimal schedule is all the b-jobs with gain m^3. In any case, the competitive ratio of $\mathcal{A}$ on S would be m.

However, the random choices of $\mathcal{A}$ are not known. Instead, the adversary will, at the onset of any time unit $t+1$, compute $p_a(t)$, the probability that $\mathcal{A}$ still serves the a-job. If $p_a(t)$ is large enough (i.e., above some threshold described below) then another b-job is requested. Otherwise the request sequence is stopped. The threshold is computed as follows. Let $\mathcal{O}_b(t)$ denote the maximum gain that can be accrued from the requested b-jobs up to time t (that is, $\mathcal{O}_b(t) = t \cdot m$ for time t where $p_a(t)$ has not yet dropped below the threshold). Let the threshold at time t be $f(\mathcal{O}_b(t))$ (that is, $f(t \cdot m)$), where the *threshold function* $f(\cdot)$ is:

$$f(x) = \begin{cases} 1 - \alpha \mathrm{e}^{\frac{x}{m^2}} & \text{if } x \le m^2, \\ 1 - \alpha \mathrm{e} & \text{if } x > m^2 \ , \end{cases}$$

and $\alpha = \frac{1}{\mathrm{e}+1}$. The adversarial strategy can now be described as follows. `First, request an` a`-job and a` b`-job at time 0. Next, at each time` $t = 1, \ldots, m^2 - 1$ `do: if` $p_a(t-1) \ge f((t-1)m)$ `request a` b`-job; else end the request sequence.`

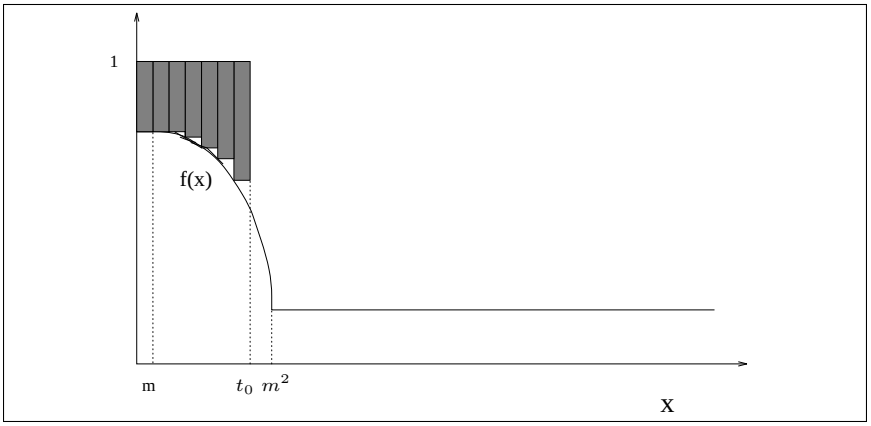


FIG. 3.2. *A* 1*-adversary. The shaded area shows the expected gain of the algorithm from type-b jobs.*

We suggest the following explanation for our choice of the threshold function. Let t_0 be the first time that $p_a(t)$ drops below the threshold. It will be seen that the competitive ratio of the algorithm is roughly at least the reciprocal of

$$f(x) + \frac{1}{m^2}\int_0^x (1 - f(y))dy \quad \text{if } x \le m^2, \tag{3.1}$$

$$1 - f(x) \quad \text{if } x > m^2\ , \tag{3.2}$$

where $x = t_0 \cdot m$ and $f(\cdot)$ is the threshold function in use (see Figure 3.2). This choice of $f(x)$ ensures that expression (3.1) evaluates to the same minimal value for all x. In particular, for any α the function $f(x) = 1 - \alpha e^{\frac{x}{m^2}}$ solves the differential equation

$$\frac{d}{dx}\left[f(x) + \frac{1}{m^2}\int_0^x (1 - f(y))dy\right] = 0,$$

thus making sure that (3.1) doesn't change for $0 \le x \le m^2$. The choice of α is such that $1 - \alpha = \alpha e$; that is, $f(0) = f(m^2) = 1 - f(x)$ for all $x > m^2$. Thus (3.1) has the same value also for $x > m^2$. Consequently, the competitive ratio of $\mathcal{A}$ on S will be roughly the same (minimal) value for all values of t_0.

Analysis of the operation of $\mathcal{A}$ on S. We consider three cases.

Case 1. $t_0 \le m$. In this case the best feasible subsequence of S is the a-job, with gain m^2 (that is, $\mathcal{O}(S) = m^2$). The expected gain of $\mathcal{A}$ is computed as follows. The probability that $\mathcal{A}$ completes the a-job is $p_a(t_0)$. The probability that $\mathcal{A}$ completes the b-job offered at time t is $p_b(t) \le 1 - p_a(t)$. Using the observation from the end of section 2, we have

$$\mathrm{E}G\mathcal{A}(S) \le m^2 p_a(t_0) + m\sum_{t=0}^{t_0}(1 - p_a(t)).$$

It follows from the adversary strategy that $p_a(t_0) < f(t_0 \cdot m)$, and $p_a(t) \ge f(t \cdot m)$ for all $t < t_0$. Thus,

$$\mathrm{E}G\mathcal{A}(S) \le m^2 f(t_0 \cdot m) + m\left(\sum_{t=0}^{t_0-1}(1 - f(t \cdot m))\right) + m \cdot p_b(t_0) \tag{3.3}$$

$$\le m^2 f(t_0 \cdot m) + \int_0^{t_0 m}(1 - f(x))dx + m \tag{3.4}$$

$$= m^2(1 - \alpha \mathrm{e}^{\frac{t_0}{m}}) + \int_0^{t_0 m} \alpha \mathrm{e}^{\frac{x}{m^2}} dx + m \tag{3.5}$$

$$= m^2(1 - \alpha \mathrm{e}^{\frac{t_0}{m}}) + m^2\alpha(\mathrm{e}^{\frac{t_0}{m}} - 1) + m \tag{3.6}$$

$$= m^2(1 - \alpha) + m. \tag{3.7}$$

(See Figure 3.2.) Thus, in this case the competitive ratio of $\mathcal{A}$ is at best $\frac{m^2}{m^2(1-\alpha)+m} = \frac{1}{1-\alpha} - O(\frac{1}{m})$.

Case 2. $m < t_0 < m^2 - 1$. In this case the best feasible subsequence of S is all the b-jobs, with gain $t_0 m$. For the expected gain of $\mathcal{A}$, we now have

$$\begin{aligned}
\mathrm{E}G\mathcal{A}(S) &\le m^2 f(t_0 \cdot m) + m\left(\sum_{t=0}^{t_0-1}(1 - f(t \cdot m))\right) + m \cdot p_b(t_0) \\
&\le m^2(1 - \alpha\mathrm{e}) + m\left(\sum_{t=0}^{m-1} \alpha \mathrm{e}^{\frac{t}{m}}\right) + m\left(\sum_{t=m}^{t_0-1} \alpha\mathrm{e}\right) + m \\
&\le \left[m^2(1 - \alpha\mathrm{e}) + \int_0^{m^2} \alpha \mathrm{e}^{\frac{x}{m^2}} dx\right] + (t_0 m - m^2)\alpha e + m \\
&\le m^2(1 - \alpha) + (t_0 m - m^2)\alpha e + m \\
&\le t_0 m \alpha \mathrm{e} + m.
\end{aligned}$$

(The derivation of the third row from the second is done using the same integral as in (3.4)–(3.7); the derivation of the fifth row from the fourth uses the fact that $1 - \alpha = \alpha\mathrm{e}$.) Thus, in this case the competitive ratio of $\mathcal{A}$ is at least $\frac{1}{\alpha\mathrm{e}} - O(\frac{1}{m})$.

Case 3. $t_0 = m^2 - 1$. The difference from Case 2 is that here $p_a(t)$ may never go below the threshold. However, this does not help the scheduler. The best feasible subsequence of S is all the b-jobs, with gain m^3. Bounding the expected gain of the scheduler, we have (since $p_a(t) > 1 - \alpha\mathrm{e}$ for all t):

$$\mathrm{E}G\mathcal{A}(S) \le m^2 p_a(m^2 - 1) + m \sum_{t=1}^{m^2-1}(1 - p_a(t)) \le m^2 + m^3(\alpha\mathrm{e}).$$

Thus, the bound on the competitive ratio of $\mathcal{A}$ is the same as in the previous case.

In all three cases, the competitive ratio of $\mathcal{A}$ on S is at least $\min\left\{\frac{1}{1-\alpha}, \frac{1}{\alpha\mathrm{e}}\right\} - O(\frac{1}{m}) = \frac{\mathrm{e}+1}{\mathrm{e}} - O(\frac{1}{m})$. A straightforward calculation shows that any $m \ge 8$ is enough for showing a nontrivial bound (that is, a bound more than 1).

3.2. Proof of Theorem 1.

Outline. Roughly speaking, the scheme described in section 3.1 used the b-jobs to make sure that $\mathcal{A}$ accrues only a fraction of the value of the a-job requested. The same scheme can be used recursively to make sure that $\mathcal{A}$ accrues only a fraction of the value of each b-job requested. That is, use a third type of jobs, called c-jobs, with smaller duration and value than b-jobs. The c-jobs have the property, however, that the total

TABLE 3.1
Summary of main notation.

Notation	Meaning	Page
i-job	a job of duration $d_i = k^{4i}$ and value $v_i = k^{2+2i}$	1002
i-period	the time period between two consecutive multiples of d_i	1002
i-small job	a j-job with $j < i$	1002
$\mathcal{O}_i(t)$	off-line gain from i-small jobs in this i-period until time t	1002
$p_i(t)$	Prob(a j-job is running at time t)	1002
$q_i(t)$	Prob(a j-job is running at time $t \mid$ no higher index job is running)	1002
i-step	the time interval between two subsequent increases in $\mathcal{O}_{i+1}(t)$	1005
i-height of τ	the amount of increase in $\mathcal{O}_{i+1}(t)$ during i-step τ	1005
$EG\mathcal{A}(S)$	the expected gain of $\mathcal{A}$ from sequence S	997
$EG\mathcal{A}_\tau(S)$	the expected gain of $\mathcal{A}$ from jobs in S requested during step τ	1005

value of a series of consecutive c-jobs scheduled back-to-back for the duration of a single b-job is greater than the value of the b-job. For each requested b-job recursively apply the above scheme, using the c-jobs. Consequently, the "effective gain" that the scheduler accrues from each b-job is only a fraction of its real value. Thus, a better threshold function can now be used with respect to the a-job, forcing $\mathcal{A}$ to accrue only a smaller fraction of the optimal gain of the entire sequence. The adversary used for proving Theorem 2 uses this idea, implementing several levels of recursion. Throughout the proof we introduce various notation. Table 3.1, summarizing the main definitions, will hopefully facilitate the reading.

Construction of a k-adversary. Define ADV_k, an adversary implementing k levels of recursion, as follows. ADV_k requests $k+1$ different types of jobs. The value of a job of type i (where $i = 0, \ldots, k$) is $v_i = k^{2+2i}$ and its duration is $d_i = k^{4i}$. Thus, $\frac{v_{i+1}}{v_i} = k^2$. (Using the terms of section 3.1, the lower index jobs play the role of the b-jobs and the higher index jobs play the role of the a-jobs.) Let an *i-period* be the time period in between two consecutive integer multiples of d_i. A job is said to be *i-small* if it is a j-job with $j < i$. Let the *partial gain function* $\mathcal{O}_i(t)$ denote the maximum (off-line) gain that can be obtained by scheduling only i-small jobs that have been requested by the adversary since the beginning of the current i-period through time t. ADV_k employs k threshold functions $f_1(\cdot), \ldots, f_k(\cdot)$, defined below. The ith threshold at time t is $f_i(\mathcal{O}_i(t))$.

At each time t, the threshold values are compared with the following values, derived from the behavior of the scheduler. Recall that $p_i(t)$ denotes the probability that the scheduler has a job of type i running at time t. Let $q_k(t) = p_k(t)$; for $i \le k$, let $q_i(t)$ denote the probability that the scheduler has a job of type i running at time t, conditioned on the event that no j-job with $j > i$ is currently running. That is,

$$q_i(t) = \frac{p_i(t)}{1 - \sum_{j=i+1}^{k} p_j(t)}.$$

ADV_k thus operates as follows. At time $t = 0$, request $k+1$ jobs, one of each type. At each other time $t+1$, run procedure $\text{ADVSTRATEGY}(k, q_k(t))$, described in Figure 3.3 below.

Let the ith threshold function be:

$$f_i(x) = \begin{cases} 1 - \alpha_i e^{\frac{\beta_{i-1}}{v_i} \cdot x} & \text{if } x \le v_i, \\ 1 - \alpha_i e^{\beta_{i-1}} & \text{if } x > v_i \ , \end{cases}$$

ADVSTRATEGY$(i, q_i(t))$:
(1) **if** d_i divides t **then**

Request an i-job;
(2) **if** $i > 0$ and $q_i(t) \geq f_i(\mathcal{O}_i(t))$ **then**
call ADVSTRATEGY$(i-1, q_{i-1}(t))$
end

FIG. 3.3. *At each time unit t,* ADV$_k$ *runs* ADVSTRATEGY$(k, q_k(t))$.

where β_i and α_i are defined as follows. Let $\beta_0 = 1$ and

$$\beta_i = \beta_{i-1}\left[\frac{e^{\beta_{i-1}}}{1+\beta_{i-1}\cdot e^{\beta_{i-1}}}\right] + \gamma, \tag{3.8}$$

where $\gamma = \frac{2}{k^2}$. Let $\alpha_i = 1 - (\beta_i - \gamma) = \frac{1}{1+\beta_{i-1}\cdot e^{\beta_{i-1}}}$. Note that the 1-adversary of section 3.1 is identical to ADV$_1$, with $v_0 = m = k^2$ and $v_1 = m^2$.

Some intuition. We suggest the following explanation to the operation of the adversary. The condition in step (1) makes sure that the times at which an i-job may be requested are d_i time units apart. Thus, only a single i-job can be requested during an i-period, and the duration of an i-job is a full i-period. (Namely, i-jobs are requested "back-to-back.") The condition in step (2) of each level i controls whether to call the $(i-1)$st level. Roughly, the $(i-1)$st level is called if the probability that an i-job is being served by $\mathcal{A}$ at time t, given that all the j-jobs for $j = i+1, \ldots, k$ were preempted, is more than the threshold. This condition corresponds to the condition of section 3.1 regarding whether to request any further b-jobs.

Also here, the threshold functions are chosen so that in each level i the scheduler will have the same expected gain regardless of when $q_i(t)$ "dips" below the threshold. The competitive ratio of the scheduler against ADV$_i$ is later shown to be roughly bounded by the reciprocal of

$$f_i(x) + \beta_{i-1}\frac{1}{v_i}\int_0^x (1 - f_i(y))dy \quad \text{if } x \leq v_i, \tag{3.9}$$

$$\beta_{i-1}(1 - f_i(v_i)) \quad \text{if } x > v_i\ , \tag{3.10}$$

where $f_i(\cdot)$ is the ith threshold function in use, and $x = \mathcal{O}_i(t)$. (The value $1/\beta_i$ represents a lower bound on the competitive ratio of a scheduler against ADV$_i$. See Lemma 1 below.) For every α_i and β_i, the choice of $f_i(x)$ ensures that expression (3.9) evaluates to the same (minimal) value for all $x \leq v_i$, using a differential equation in the same way as in section 3.1. The α_i's and β_i's are determined as follows. We set $\beta_0 = 1$. Next, for any $i > 0$, α_i is chosen so that the competitive ratios at $x \leq v_i$ and at $x > v_i$ are equal; using (3.9) and (3.10) this translates to $f_i(0) = \beta_{i-1}(1 - f_i(v_i))$, or (by the definition of $f_i(x)$):

$$\alpha_i = \frac{1}{\beta_{i-1}\mathrm{e}^{\beta_{i-1}} + 1} \tag{3.11}$$

(for $i = 1$ this condition identifies with the corresponding condition in section 3.1: $\alpha_1 = \frac{1}{\mathrm{e}+1}$). Next, β_i can be computed by evaluating (3.9) at any value of x (and adding a small "error term" γ). Setting, say, $x = 0$ and using (3.11), we get $\beta_i = f_i(0) + \gamma = 1 - \alpha_i + \gamma = 1 - \frac{1}{\beta_{i-1}\mathrm{e}^{\beta_{i-1}}+1} + \gamma$. The recursion relation (3.8) follows.

Analysis of ADV_k.

LEMMA 1. *Let $k \geq 0$. Let $\mathcal{A}$ be a scheduler and let S be the request sequence generated via an interaction between $\mathcal{A}$ and ADV_k. Then, the competitive ratio of $\mathcal{A}$ on S is at best $\frac{1}{\beta_k+\delta}$, where $\delta \stackrel{\triangle}{=} \frac{3}{k-1}$.*

In order to derive Theorem 2 from Lemma 1, we need the following technical lemma.

LEMMA 2. *Let β_k be as defined in* (3.8). *Then $\beta_k \leq 4/\sqrt{k}$.*

Proof. We will verify inductively that

$$\beta_i \leq 4/\sqrt{i} \quad \text{for } i \leq k. \tag{3.12}$$

Plugging $\beta_0 = 1$ into (3.8) with $i = 1$, we get that $\beta_1 \leq \frac{3}{4}$. Since β_i decreases with i, we know that (3.12) holds whenever $3/4 \leq 4/\sqrt{i}$. Thus, we only have to worry about the cases where $3/4 > 4/\sqrt{i}$. Since the function

$$F(x) = x\left[\frac{e^x}{1+xe^x}\right]$$

is increasing in x, we can use the upper bound on β_i from the inductive hypothesis to obtain an upper bound for β_{i+1} as follows:

$$\frac{4}{\sqrt{i}}\left[\frac{e^{\frac{4}{\sqrt{i}}}}{1+\frac{4}{\sqrt{i}}e^{\frac{4}{\sqrt{i}}}}\right] + \gamma \geq \beta_i\left[\frac{e^{\beta_i}}{1+\beta_i \cdot e^{\beta_i}}\right] + \gamma \geq \beta_{i+1}.$$

Plugging in $\gamma = \frac{2}{k^2} \geq \frac{2}{i^2}$, it just has to be verified that

$$\frac{4}{\sqrt{i+1}} \geq \frac{4}{\sqrt{i}}\left[\frac{e^{\frac{4}{\sqrt{i}}}}{1+\frac{4}{\sqrt{i}}e^{\frac{4}{\sqrt{i}}}}\right] + \frac{2}{i^2}.$$

After some algebraic manipulation, it can be seen that this is equivalent to

$$\frac{i}{i+1} - \frac{1}{i^{\frac{3}{2}}} + \frac{1}{4i^3} \geq e^{\frac{8}{\sqrt{i}}}\left(1 - \frac{4}{\sqrt{i+1}} + \frac{2}{i^2}\right)^2.$$

From the Taylor expansion of e^x, we can verify that $e^x \leq (1+x+\frac{3}{4}x^2)$ for any $x \leq 1$. Using the fact that $4/\sqrt{i} < 3/4$, we can assume that $i \geq 28$. Using both of these facts, it is not hard to verify that $4/\sqrt{i} - 1/7 \leq 4/\sqrt{i+1}$. Plugging all of these back into the above inequality establishes (3.12). ∎

Now, to establish Theorem 2, we first compute the maximum possible value for k (namely, the maximum possible number of recursion levels), and we note that a sequence S generated by ADV_k has $\mu_v(S) = \frac{v_0}{v_k} = k^{2k}$ (and $\mu_d(S) = \frac{v_k}{v_o} = k^{4k}$). Thus, given that $\mu(S) = m$ we can employ ADV_k, where $k = \lfloor \frac{\log m}{4 \log\log m} \rfloor$.

Theorem 2 always holds for $\sqrt{\frac{\log m}{\log\log m}} \leq 10$, since the competitive ratio is always at least 1. Thus, if we use $k = \lfloor \frac{\log m}{4\log\log m} \rfloor$, we can assume that $k \geq 25$. Using this lower bound on k, the fact that $\delta = \frac{3}{k-1}$ and the result from Lemma 2, we can bound $\beta_k + \delta$ by $\frac{5}{\sqrt{k}}$. Using Lemma 1 and plugging in $k = \lfloor \frac{\log m}{4\log\log m} \rfloor$, establishes Theorem 2. ∎

Proof of Lemma 1. Consider a sequence S generated via an interaction between $\mathcal{A}$ and ADV_k. We introduce the following notation. A time unit t is called *i-critical* if $\mathcal{O}_{i+1}(t) > \mathcal{O}_{i+1}(t-1)$. A time interval $\tau = [t_s, t_f]$ is called an i-step if t_s and $t_f + 1$ are i-critical and $t_s+1, \ldots, t_f$ are not i-critical. For every i, the sequence S partitions time into a sequence of i-steps. Let the *i-height* of τ be $h_i(\tau) = \mathcal{O}_{i+1}(t_s) - \mathcal{O}_{i+1}(t_s - 1)$. (Note that if $\tau = [t_s, t_f]$ is the first i-step in an i-period then the i-height of τ is v_i.) The optimal gain from sequence S is the sum of the k-heights of all the k-steps in the duration of S.

Roughly, we will show that the expected gain of $\mathcal{A}$ from the jobs requested during each k-step is at most β_k times the k-height of this step. However, we first distinguish between the main contribution to the competitive ratio of $\mathcal{A}$ and several special cases which result in additional, small "error terms." These error terms, encapsuled in the term δ defined above, result from three classes of requests. These classes are defined below and are taken care of in Lemma 4.

A j-job c is said to be *final for level i* if $j < i$ and c is the last j-job to be requested in its i-period. A j-job is *final* if it is final for some level i where $i > j$. A j-job c is a *step-2 job for level i* if $i > j$ and c is requested during the *second* i-step in its i-period. A job is *step-2* if it is step-2 for some level i with $i > j$. A j-job c is said to be *high-probability* if $q_j(t) \geq f_j(\mathcal{O}_j(t))$ for all times t in its duration. (High probability jobs correspond, in principle, to Case 3 in the analysis of a 1-adversary in section 3.1. This is the case where the probability that a job is running never falls below the threshold.) A job is *regular* if it is neither a step-2, final, nor high-probability job.

Let $\text{E}G\mathcal{A}_\tau(S)$ denote the expected gain of $\mathcal{A}$ from the *regular* jobs requested by ADV_k during some step τ. That is, $\text{E}G\mathcal{A}_\tau(S) = \sum_{c \in S_\tau} v_c \cdot \text{Prob}\ (\mathcal{A}\ \text{completes}\ c)$, where S_τ is the sequence of regular jobs requested during τ. Lemma 3 states that $\text{E}G\mathcal{A}_\tau(S)$ is at most β_k times the k-height of each k-step τ. We know that $\mathcal{O}(S) = \sum_{\{k\text{-steps } \tau\}} h_k(\tau)$. It follows that the expected gain of $\mathcal{A}$ from S due to regular jobs is at most $\beta_k \cdot \mathcal{O}(S)$. Lemma 4 states that the total gain of $\mathcal{A}$ from all the nonregular jobs in S is at most $\delta \cdot \mathcal{O}(S)$. Lemma 1 follows. □

On the structure of S. In the proofs of Lemmas 3 and 4, we use the following observations regarding the structure of S. The first job requested in each i-period is an i-job. The rest of the jobs in this i-period are i-small. Each i-period can be partitioned into i-steps (any i-step is contained within a single i-period). The first i-step in an i-period occurs due to the request of the i-job and has i-height v_i. This i-step can be partitioned to several $(i-1)$-steps. In the second i-step (if there is one), the optimal gain due to the i-small jobs exceeds the gain of the i-job for the first time. If this happens, then the optimal schedule in the i-period contains only i-small jobs instead of the i-job. As a result, any additional increase in $\mathcal{O}_{i+1}$ within this i-period causes the same increase in $\mathcal{O}_i$, and any subsequent i-step τ is also an $(i-1)$-step. Furthermore, $h_i(\tau) = h_{i-1}(\tau)$. Note that if there is a second i-step τ, then τ is also an $(i-1)$-step, but it is not necessarily the case that $h_i(\tau) = h_{i-1}(\tau)$ for the second i-step. This is because the arrival of an i-small job may cause the optimal gain from i-small jobs to exceed the value of the i-job by only a small amount.

LEMMA 3. *Let $k \geq 0$. Let $\mathcal{A}$ be a scheduler and let S be the request sequence generated via an interaction between $\mathcal{A}$ and ADV_k. Let τ be a k-step. Then,*

$$\text{E}G\mathcal{A}_\tau(S) \leq \beta_k \cdot h_k(\tau). \tag{3.13}$$

Proof. Outline. We prove the lemma by induction on k. The lemma trivially holds for ADV_0 (using one type of job), since $\beta_0 = 1$. Also, the case of ADV_1 was analyzed in section 3.1.

Let $k > 0$ and assume that there exists a scheduler $\mathcal{A}$ and a k-step τ such that (3.13) is violated. We construct a scheduler $\mathcal{A}'$ with the following property that contradicts the induction hypothesis. Consider the sequence S' generated via the interaction of $\mathcal{A}'$ with ADV_{k-1}. Then there exists a $(k-1)$-step τ' in S' such that

$$\text{EG}\mathcal{A}'_{\tau'}(S') > \beta_{k-1} \cdot h_{k-1}(\tau'), \tag{3.14}$$

where $\text{EG}\mathcal{A}'_{\tau'}(S')$ is defined similarly to $\text{EG}\mathcal{A}_\tau(S)$, with respect to $\mathcal{A}'$, S', and τ'.

Construction of $\mathcal{A}'$. Consider the sequence S generated by ADV_k interacting with $\mathcal{A}$. First a $(k-1)$-period of S, starting at time t_*, is chosen in a way described below. Roughly, $\mathcal{A}'$ will imitate the operation of $\mathcal{A}$ starting at time t_*, conditioned on the event that $\mathcal{A}$ has preempted the k-job. More precisely, let $p_i(t)$ (resp., $p'_i(t)$) denote the probability that $\mathcal{A}$ (resp., $\mathcal{A}'$) has a job of type i scheduled at time unit t. Scheduler $\mathcal{A}'$ is constructed so that when playing against ADV_{k-1} for the duration of a $(k-1)$-period, the following probability vector $\vec{p}'(t) = p'_0(t), \ldots, p'_{k-1}(t)$ is maintained:

$$p'_i(t) = \frac{p_i(t+t_*)}{1 - p_k(t+t_*)}. \tag{3.15}$$

Scheduler $\mathcal{A}'$ can always be implemented since $\sum_{i=0}^{k-1} p'_i(t) \le 1$, and $p'_i(t) > p'_i(t-1)$ only if an i-job is requested at time t. In choosing t_* we distinguish two cases.

Case 1. Step τ is not the first k-step. Then τ is also a $(k-1)$-step and is contained in a $(k-1)$-period. Let t_* be the beginning of this $(k-1)$-period. In the analysis, we will divide this case into two subcases, depending on whether the step is the second k-step or not.

Case 2. Step τ is the first k-step. The first k-step is partitioned into several $(k-1)$-steps, $\tau_1, \ldots, \tau_l$. Let τ_* be the $(k-1)$-step contained in τ that maximizes the ratio

$$\max_{1 \le i \le l} \left\{ \frac{\text{EG}\mathcal{A}_{\tau_i}(S)}{(1 - p_k(e_{\tau_i})) \cdot h_{k-1}(\tau_i)} \right\}$$

among all the $(k-1)$-steps contained in τ, where e_{τ_i} is the latest ending time of a regular job requested during τ_i (note that e_{τ_i} may be later than the end of τ_i). For convenience, we assume that the initial k-job is not included in the first $(k-1)$-step. Let t_* be the beginning of the $(k-1)$-period containing τ_*. Note that $\mathcal{A}'$ may perform very poorly on request sequences different than those generated by ADV_{k-1}. The only purpose of $\mathcal{A}'$ is to contradict the induction hypothesis.

Analysis of scheduler $\mathcal{A}'$. It can be seen from (3.15) and the adversary strategy (Figure 3.3) that at each time t in the interaction of ADV_{k-1} with $\mathcal{A}'$, ADV_{k-1} requests exactly the same jobs that ADV_k requests at time $t + t_*$ when interacting with $\mathcal{A}$; this holds with the following two exceptions. First, ADV_k may request an initial k-job at time t_*, where ADV_{k-1} does not. Second, the sequence S may end before S', if $p_k(t)$ falls below the threshold for some time t. Let $\hat{S}$ denote the subsequence of S starting at time t_* with the initial k-job removed (if there is one). Then, $\hat{S}$ is a prefix of S'. Furthermore, for any $(k-1)$-step τ in $\hat{S}$ we have $h_{k-1}(\tau) = h_{k-1}(\tau')$, where τ' is the $(k-1)$-step in S' that corresponds to τ. We consider the above two cases of determining t_*.

Case 1a. Step τ is the second k-step. All jobs requested in this step are step-2 jobs and are thus not regular. Thus, $\mathrm{E}G\mathcal{A}_\tau(S) = 0$. (Step-2 jobs are accounted for in Lemma 4.)

Case 1b. Step τ is neither the first nor the second k-step. Then, τ is a $(k-1)$-step as well. Let τ' be the $(k-1)$-step that corresponds to τ in S'. Then, $h_k(\tau) = h_{k-1}(\tau) = h_{k-1}(\tau')$. We show below that the probability that $\mathcal{A}$ completes each regular job $c \in S$ is at most $\frac{\beta_k}{\beta_{k-1}}$ times the probability that $\mathcal{A}'$ completes the corresponding job $c' \in S'$ during τ'. Thus, if (3.13) is violated with respect to τ then (3.14) holds with respect to step τ'.

Scheduler $\mathcal{A}$ completes an i-job c with probability $p_i(e_c)$, and $\mathcal{A}'$ completes c' with probability $\frac{p_i(e_c)}{1-p_k(e_c)}$, where e_c is the time at which c ends. It remains to show that $1 - p_k(e_c) \le \frac{\beta_k}{\beta_{k-1}}$. Since c is not a final job, we have that $p_k(e_c) \ge f_k(\mathcal{O}_k(e_c))$. Since τ is not the first k-step, the optimal gain has already exceeded v_k, and $\mathcal{O}_k(e_c) > v_k$. Thus, $f_k(\mathcal{O}_k(e_c)) = 1 - \alpha_k \mathrm{e}^{\beta_{k-1}}$ and $1 - p_k(e_c) \le \alpha_k \mathrm{e}^{\beta_{k-1}} \le \frac{\beta_k}{\beta_{k-1}}$.

Case 2. Step τ is the first k-step. Let ${\tau_*}'$ be the $(k-1)$-step that corresponds to τ_* with respect to S'. We show that if $\mathrm{E}G\mathcal{A}'_{{\tau_*}'}(S') \le \beta_{k-1} \cdot h_{k-1}({\tau_*}')$ then $\mathrm{E}G\mathcal{A}_\tau(S) \le \beta_k \cdot v_k$. We proceed in two steps: first we show (3.17), where t_0 is defined below. Next we show (3.18).

Showing (3.17), we first express $\mathrm{E}G\mathcal{A}_\tau(S)$ in terms of the expected gain of $\mathcal{A}$ in the $(k-1)$-steps $\tau_1, \ldots, \tau_l$ contained in τ. Assume that the initial k-job is not a high-probability job. In this case, there is a time t such that $p_k(t) < f_k(\mathcal{O}_k(t))$. Let $t_0 = t$ if t happens before the end of τ; otherwise t_0 is the first time unit after the end of τ. Thus,

$$\mathrm{E}G\mathcal{A}_\tau(S) \le v_k \cdot f_k(\mathcal{O}_k(t_0)) + \sum_{i=1}^{l} \mathrm{E}G\mathcal{A}_{\tau_i}(S). \tag{3.16}$$

Inequality (3.16) is satisfied even if the initial k-job is a high-probability job. In this case, the contribution of this job is not included in $\mathrm{E}G\mathcal{A}_\tau(S)$; it is considered in Lemma 4.

Scheduler $\mathcal{A}'$ completes each job $c' \in S'$ requested during τ_* with probability at least $\frac{1}{(1-p_k(e_c))}$ times the probability that $\mathcal{A}$ completes the corresponding job $c \in S$. Since $e_c \le e_{\tau_*}$, we have $\mathrm{E}G\mathcal{A}_{\tau_*}(S) \le (1 - p_k(e_{\tau_*})) \cdot \mathrm{E}G\mathcal{A}'_{{\tau_*}'}(S')$. Recall that $\mathrm{E}G\mathcal{A}'_{{\tau_*}'}(S') \le \beta_{k-1} \cdot h_{k-1}({\tau_*}')$, and $h_{k-1}({\tau_*}') = h_{k-1}(\tau_*)$. Thus,

$$\mathrm{E}G\mathcal{A}_{\tau_*}(S) \le (1 - p_k(e_{\tau_*})) \cdot \beta_{k-1} \cdot h_{k-1}(\tau_*).$$

It follows from the choice of τ_* that $\mathrm{E}G\mathcal{A}_{\tau_i}(S) \le (1 - p_k(e_{\tau_i}))\beta_{k-1} \cdot h_{k-1}(\tau_i)$ for all steps τ_i. Thus, (3.16) implies that

$$\mathrm{E}G\mathcal{A}_\tau(S) \le v_k \cdot f_k(\mathcal{O}_k(t_0)) + \sum_{i=1}^{l} (1 - p_k(e_{\tau_i})) \cdot \beta_{k-1} \cdot h_{k-1}(\tau_i). \tag{3.17}$$

It remains to show that

$$v_k \cdot f_k(\mathcal{O}_k(t_0)) + \sum_{i=1}^{l} (1 - p_k(e_{\tau_i})) \cdot \beta_{k-1} \cdot h_{k-1}(\tau_i) \;\le\; \beta_k \cdot v_k. \tag{3.18}$$

First, we show that

$$v_k \cdot f_k(\mathcal{O}_k(t_0)) + \sum_{i=1}^{l} (1 - p_k(s_{\tau_i})) \cdot \beta_{k-1} \cdot h_{k-1}(\tau_i) \le (\beta_k - \gamma) \cdot v_k, \tag{3.19}$$

where s_{τ_i} is the starting time of step τ_i, and $\gamma = \frac{2}{k^2}$ is a small "error term." Next we show that

$$\sum_{i=1}^{l}(p_k(s_{\tau_i}) - p_k(e_{\tau_i})) \cdot \beta_{k-1} \cdot h_{k-1}(\tau_i) \le \gamma \cdot v_k. \tag{3.20}$$

Showing (3.19), we have

$$\begin{aligned}
&v_k \cdot f_k(\mathcal{O}_k(t_0)) + \sum_{i=1}^{l}(1 - p_k(s_{\tau_i})) \cdot \beta_{k-1} \cdot h_{k-1}(\tau_i) \\
&\le v_k \cdot f_k(\mathcal{O}_k(t_0)) + \beta_{k-1} \cdot \sum_{i=1}^{l}(1 - f_k(\mathcal{O}_k(s_{\tau_i}))) \cdot h_{k-1}(\tau_i) \\
&\le v_k \cdot f_k(\mathcal{O}_k(t_0)) + \beta_{k-1} \cdot \int_0^{\mathcal{O}_k(t_0)}(1 - f_k(y))dy = (*).
\end{aligned}$$

The last derivation uses a standard transformation from a sum to an integral. We integrate only up to $\mathcal{O}_k(t_0)$, since either t_0 is after the end of τ, or the only jobs requested at or after time t_0 are final jobs. Let $x = \mathcal{O}_k(t_0)$. Since this is the first k-step we have $x \le v_k$; thus, $f_k(x) = 1 - \alpha_k e^{\frac{\beta_{k-1}x}{v_k}}$. Thus:

$$\begin{aligned}
(*) &= v_k \cdot f_k(x) + \beta_{k-1} \cdot \int_0^x (1 - f_k(y))dy \\
&= \left(1 - \alpha_k e^{\frac{\beta_{k-1}x}{v_k}}\right) v_k + \beta_{k-1} \int_0^x \alpha_k e^{\frac{\beta_{k-1}y}{v_k}} dy \\
&= \left(1 - \alpha_k e^{\frac{\beta_{k-1}x}{v_k}}\right) v_k + \beta_{k-1} \frac{v_k}{\beta_{k-1}} \alpha_k \left(e^{\frac{\beta_{k-1}x}{v_k}} - 1\right) \\
&= (1 - \alpha_k)v_k \\
&= (\beta_k - \gamma)v_k.
\end{aligned}$$

To show (3.20), let w_i denote the largest j such that a j-job is requested on the first time unit of τ_i. Rewriting the left-hand side of (3.20) we get

$$\beta_{k-1}\sum_{i=1}^{l} h_{k-1}(\tau_i)\cdot(p_k(s_{\tau_i})-p_k(e_{\tau_i})) \le \beta_{k-1}\sum_{j=0}^{k-1} v_j \cdot \sum_{\{i\,:\,w_i=j\}}(p_k(s_{\tau_i})-p_k(e_{\tau_i})) = (**).$$

Observe that $p_k(t)$ is a nonincreasing function, and that for any w_{i_1}, w_{i_2} with $i_1 > i_2$, step τ_{i_1} begins only after all the jobs requested during τ_{i_2} have ended. Thus, $\sum_{\{i \mid w_i=j\}}(p_k(s_{\tau_i}) - p_k(e_{\tau_i})) \le 1$. Since $v_{w_i} = k^{2+2w_i}$ and $\beta_{k-1} \le 1$, we have

$$(**) \le \beta_{k-1}\sum_{j=0}^{k-1} v_j \cdot 1 \le 2 \cdot v_{k-1}\cdot \le \frac{2}{k^2} \cdot v_k = \gamma \cdot v_k. \qquad \square$$

LEMMA 4. *Let $k \ge 0$. Let $\mathcal{A}$ be a scheduler and let S be the request sequence generated by an interaction between $\mathcal{A}$ and* ADV$_k$. *Then, the total gain of $\mathcal{A}$ from all the step-2, final, and high-probability jobs in S is at most $\frac{3}{k-1}\mathcal{O}(S)$.*

Proof. We first bound the gain of $\mathcal{A}$ from the final and step-2 jobs. The gain achievable, even by the optimal schedule, from all the step-2 jobs for level i requested within a given i-period is at most v_{i-1}. For each i-job and every $j < i$, there is at most one final j-job. Let n_i be the total number of i-jobs in S. The optimal gain achievable from the jobs that are either final or step-2 for level i is thus at most

$$n_i \left(v_{i-1} + \sum_{j=0}^{i-1} v_j \right) \leq 2n_i v_i \sum_{j=0}^{i-1} \frac{1}{k^{2(i-j)}} \leq \frac{2n_i v_i}{k^2 - 1}.$$

For every i, the sequence of all i-jobs in S is a feasible subsequence of S, with gain $n_i v_i$. Therefore, $\mathcal{O}(S) \geq \max_i(n_i v_i)$. The sum of the values of all final and step-2 jobs is thus at most:

$$\sum_{i=0}^{k} \frac{2n_i v_i}{k^2 - 1} \leq (k+1) \max_i \frac{2n_i v_i}{k^2 - 1} \leq \frac{2}{k-1} \mathcal{O}(S).$$

Next we bound the gain achievable from the high-probability jobs which are not final. Consider a high-probability i-job c which is not final. We first show that an $(i-1)$-job is requested in each $(i-1)$ period for the duration of c. To see why this is true, recall that an $(i-1)$ job is requested in an $(i-1)$-period if at the first time unit t in this period $q_j(t) \geq f_j(\mathcal{O}_j(t))$ for all $j \geq i$. If c is not final, then $q_j(t) \geq f_j(\mathcal{O}_j(t))$ for all $j > i$ and for all times t in the duration of c. If c is high-probability then $q_i(t) \geq f_i(\mathcal{O}_i(t))$ for all times t in the duration of c.

Consequently, k^4 jobs of type $(i-1)$ are requested in the duration of c. The total value of these $(i-1)$-jobs is k^2 times the value of c. Let g_i be the number of all nonfinal high-probability i-jobs in S. We know that $k^2 v_i g_i \leq v_{i-1} n_{i-1}$. Therefore, the total gain from all nonfinal high-probability jobs is at most

$$\sum_{i=1}^{k} v_i g_i \leq \frac{1}{k^2} \sum_{i=1}^{k} v_{i-1} n_{i-1} \leq \frac{1}{k} \max_i v_i n_i \leq \frac{1}{k} \mathcal{O}(S). \qquad \square$$

4. Applications of the bound. In this section, we present a series of corollaries showing how the lower bound of Theorem 2 can be generalized and extended to related problems. The proofs of most corollaries are very similar techniques. For clarity, we give a full proof of the first corollary (Corollary 1). Next we describe our general technique in abstract terms. For the rest of the proofs we merely state how certain parameters of the general technique should be set.

4.1. Multiprocessor scheduling. We first describe how the bound in Theorem 2 can be generalized to situations where more than a single job can be served at a time. Each job c has a load l_c, and a feasible sequence is one in which the sum of the loads of any set of overlapping jobs is at most the capacity U of the server. The bound holds even if there is more than one server (each with capacity U) and even if there is an upper bound on the load that any single job may require.

Recall that μ_d, μ_l, and μ_v denote the ratio of maximum to minimum duration, load and value of a job, respectively. Let $g(x) = \frac{1}{8}\sqrt{\frac{\log x}{\log \log x}}$.

COROLLARY 1. *Let $\delta < 1$. Theorem 2 holds even when there are m servers, each with capacity U, and even if $l_c \leq \delta U$ for each job c.*

Proof. We assume the existence of a scheduler $\mathcal{A}'$ that contradicts the corollary. Based on $\mathcal{A}'$, we then construct scheduler $\mathcal{A}$ that contradicts Theorem 2. $\mathcal{A}$ keeps a

copy of $\mathcal{A}'$ running. On a sequence S of requests, $\mathcal{A}$ generates for $\mathcal{A}'$ a sequence S' constructed as follows.

Construction of $\mathcal{A}$. Let α be chosen so that $1/\alpha = \lceil 1/\delta \rceil$. Upon receiving a request c with duration d_c and value v_c, $\mathcal{A}$ generates a set S'_c of $N_c = m/\alpha$ requests for $\mathcal{A}'$. Each request in S'_c has the same duration as c, has value $v_{c'} = \alpha v_c/m$ and load $l_{c'} = U\alpha$. Note that if all jobs from S'_c are scheduled, they use up the entire capacity of the system. Furthermore, the sum of the values of all the jobs in S'_c is equal to $N_c v_{c'} = v_c$.

Let $n_c(t)$ denote the expected number of jobs in S'_c scheduled by $\mathcal{A}'$ at time t. $\mathcal{A}$ will accept and preempt jobs so as to maintain the invariant that at all times t, the probability that $\mathcal{A}$ has job c scheduled is $n_c(t)/N_c$. In showing how $\mathcal{A}$ achieves this we assume that $\mathcal{A}'$ has the property that after the arrival of a set of jobs S'_c, the entire capacity of all m servers is being used. No generality is lost by this assumption, since $\mathcal{A}'$ can always decide to preempt jobs later in favor of an incoming job. Now suppose that $\mathcal{A}$ has a job $\tilde{c}$ running during time unit $t-1$ and jobs $c_1, c_2, \ldots, c_k$ arrive at time t. $\mathcal{A}$ will preempt $\tilde{c}$ with probability $(n_{\tilde{c}}(t-1) - n_{\tilde{c}}(t))/n_{\tilde{c}}(t-1)$. If $\mathcal{A}$ preempts its current job or did not have any jobs running during time $t-1$, then it accepts job c_i with probability $n_{c_i}(t)/\sum_{j=1}^{k} n_{c_j}(t)$. It can be verified by induction on t that the invariant holds.

Analysis of $\mathcal{A}$. Given a feasible subsequence $\hat{S} \subseteq S$, the following feasible subsequence $\hat{S}' \subseteq S'$ satisfies $\mathcal{O}(\hat{S}) \leq \mathcal{O}(\hat{S}')$: for every $c \in \hat{S}$, include all the jobs from S'_c in $\hat{S}'$. Thus $\mathcal{O}(S) \leq \mathcal{O}(S')$.

Let e_c be the ending time of job c. The expected gain of $\mathcal{A}'$ from the jobs in S'_c is $v_{c'} n_c(e_c) = v_c n_c(e_c)\alpha/m$. The expected gain of $\mathcal{A}$ from job c is $v_c n_c(e_c)/N_c = v_c n_c(e_c)\alpha/m$. Thus, the expected gain of $\mathcal{A}$ on its input sequence S is equal to the expected gain of $\mathcal{A}'$ on the constructed sequence S'. Since $\mu_d(S) = \mu_d(S')$ and $\mu_v(S) = \mu_v(S')$, the existence of $\mathcal{A}'$ contradicts Theorem 2. □

The remaining corollaries are proven using a reduction technique similar to the proof of Corollary 1. We first describe this technique in more general terms. Next, we state the corollaries and fill in the necessary details in the proofs.

Assume the existence of a scheduler $\mathcal{A}'$ that contradicts some corollary. Based on $\mathcal{A}'$, we then construct scheduler $\mathcal{A}$ that contradicts Theorem 2. $\mathcal{A}$ keeps a copy of $\mathcal{A}'$ running. Given a sequence S of requests, $\mathcal{A}$ gives $\mathcal{A}'$ a sequence S' constructed as follows. Upon receiving a request c in S, $\mathcal{A}$ generates a set S'_c of N_c requests for $\mathcal{A}'$, each of value $v_{c'}$ and load $l_{c'}$. The duration of each job in S'_c is exactly d_c. The parameters of the constructed requests depend on the reduction. However, they will always have the property that the total load of the N_c jobs in S'_c equals the capacity of the system. The sum of the values of all the jobs in S'_c is equal to v_c (i.e., $N_c v_{c'} = v_c$). $\mathcal{A}$ maintains the invariant that at all times t the probability that $\mathcal{A}$ has job c scheduled is $n_c(t)/N_c$. Thus, we can deduce that $\mathcal{O}(S) \leq \mathcal{O}(S')$. Also, given a feasible subsequence $\hat{S} \subseteq S$, we can obtain a feasible subsequence $\hat{S}' \subseteq S'$ such that $G(\hat{S}) \leq G(\hat{S}')$: for every $c \in \hat{S}$, include all the jobs from S'_c in $\hat{S}'$.

As in the previous proof, the expected gain of $\mathcal{A}'$ from the jobs in S'_c is $v_{c'} n_c(e_c)$. Consequently, the expected gain of $\mathcal{A}$ from job c is $v_c n_c(e_c)/N_c$.

The next two corollaries show that the bound applies also when some specific value functions are used.

COROLLARY 2. *Assume jobs have arbitrary load and duration, and the value of a job is its duration. Then, any randomized, preemptive on-line scheduler is at best* $g/2$*-competitive for measures* μ_d *and* μ_l.

Proof. We choose $N_c = v_c/d_c$, $l_{c'} = \frac{U d_c}{v_c}$, and $d_{c'} = d_c$. The expected gain of $\mathcal{A}'$ from jobs in S'_c is $d_c n_c(e_c)$, and the expected gain of $\mathcal{A}$ from job c is $v_c n_c(e_c)/N_c = d_c n_c(e_c)$. Thus, the expected gain of $\mathcal{A}$ on its input sequence S is equal to the expected gain of $\mathcal{A}'$ on the constructed sequence S'. Since $\mu_d(S) = \mu_d(S')$ and $\mu_l(S') \leq \mu_d(S) \cdot \mu_v(S)$, we have that $\max\{g(\mu_d(S')), g(\mu_l(S'))\} \leq 2\max\{g(\mu_d(S)), g(\mu_v(S))\}$. Thus, if $\mathcal{A}'$ is $g/2$-competitive with respect to measures μ_d and μ_l, then $\mathcal{A}$ is g-competitive with respect to measures μ_d and μ_v, contradicting Theorem 2.

We remark that the construction can be adapted to the cases where there is a bound δU (for any $\delta \leq 1$) on the load of a single job and where there are m servers. This is accomplished in a similar manner as in Corollary 1, that is, by increasing the number of jobs by a factor of m/δ and reducing the load of each job by a factor of δ. □

COROLLARY 3. *Assume jobs have arbitrary load and duration, and the value of all jobs is* 1. *Then, any randomized, preemptive on-line scheduler is at best g-competitive for measures μ_l and μ_d.*

Proof. We choose $N_c = v_c$, $l_{c'} = \frac{U}{v_c}$, and $d_{c'} = d_c$. The expected gain of $\mathcal{A}'$ from the S'_c jobs is $n_c(e_c)$, and the expected gain of $\mathcal{A}$ from job c is $v_c n_c(e_c)/N_c = n_c(e_c)$. Thus, the expected gain of $\mathcal{A}$ on its input sequence S is equal to the expected gain of $\mathcal{A}'$ on the constructed sequence S'. Since $\mu_d(S) = \mu_d(S')$ and $\mu_l(S') = \mu_v(S)$, the existence of $\mathcal{A}'$ contradicts Theorem 2.

Again, the construction can be adapted as in Corollary 1 to the cases where there is a bound δU (for any $\delta \leq 1$) on the load of a single job and where there are m servers. □

4.2. Bandwidth allocation. Next we address bandwidth allocation problems. Job scheduling for the uniprocessor model discussed above is identical to the bandwidth allocation (call control) model on a single link, when load is translated to bandwidth. We generalize the result to work for an arbitrary network as follows.

COROLLARY 4. *Assume calls have arbitrary values. Then, any randomized, preemptive call control algorithm for any network has a competitive ratio of at best g for measures μ_d and μ_v.*

Proof. Pick any pair of nodes s and t in the network. Let F be the value of the maximum flow from s to t, where $u(e)$ is the capacity of an edge. Let ϵ be the greatest common divisor of all the edge capacities. We will only request calls between s and t with bandwidth at most ϵ. Thus, the set of paths between s and t can be treated as a single edge with capacity F.

We use the same reduction above, choosing $N_c = F/\epsilon$, $v_{c'} = v_c\epsilon/F$, $d_{c'} = d_c$, and $l_c = \epsilon$. The expected gain of $\mathcal{A}'$ from the c' jobs is $v_c n_c(e_c)\epsilon/F$, and the expected gain of $\mathcal{A}$ from job c is $v_c n_c(e_c)/N_c = v_c n_c(e_c)\epsilon/F$. Thus, the expected gain of $\mathcal{A}$ on its input sequence S is equal to the expected gain of $\mathcal{A}'$ on the constructed sequence S'. Since $\mu_d(S) = \mu_d(S')$ and $\mu_v(S) = \mu_v(S')$, the existence of $\mathcal{A}'$ contradicts Theorem 2. □

COROLLARY 5. *Any randomized, preemptive call control algorithm for any network has a competitive ratio of at best g for measures μ_d and μ_l, even if the value of a call is its duration, or if the values of all calls are* 1.

Proof. Use the techniques of the proof of Corollary 4 to extend Corollaries 2 and 3 to general networks. □

Next we concentrate on networks with no cycles, namely trees. Corollaries 4 and 5 apply. In addition, here the bound can be shown to apply to other natural ways for determining a call value. Let r_c denote the distance in the tree between the two

endpoints of the call c. Let D be the diameter of the tree. (D can be regarded as the ratio between the lengths of the longest and shortest paths in a network.)

COROLLARY 6. *Suppose we have a tree network with diameter D. Then any randomized, preemptive call control algorithm for this network has a competitive ratio of at best $\Omega(g)$ for measures μ_d and D, even with the following criteria for call value.*

(i) $v_c = l_c d_c$.

(ii) $v_c = r_c$.

(iii) $v_c = r_c l_c$.

(iv) $v_c = r_c d_c$.

(v) $v_c = r_c l_c d_c$.

Remarks. (1) Note that although D is not a parameter of the input sequence, it describes the network which is part of the input to the problem. The definition of competitiveness in section 2 can be adapted accordingly. (2) Method (i) for determining the value of a call measures the amount of information potentially contained in a call. Measure (iv) measures the amount of "work" invested in a call. The fact that the bound holds for these measures stands in contrast to the single-edge and multiprocessor cases. There, constant competitive algorithms exist for similar measures (see, e.g., [8]).

Proof. We construct a reduction to Theorem 2. Assume that D, as well as v_c and v_c/d_c for each job c requested of $\mathcal{A}$, are powers of 2 (and d_c is an integer). Such a situation can be achieved by rescaling and rounding, with a loss of at most a factor of 4 in effectiveness. That is, first rescale v_c and d_c to be all integers. Next, D and v_c for each call c are rounded to the largest power of two smaller than D.

(i) Assume the existence of an $o(g)$-competitive scheduler $\mathcal{A}'$ on a tree network with diameter $D = \mu_d(S) \cdot \mu_v(S)$, where the value of a call c is $l_c \cdot d_c$. We construct a scheduler $\mathcal{A}$ that contradicts Theorem 2.

$\mathcal{A}$ first chooses a simple path of length D in the network on which $\mathcal{A}'$ operates and numbers the nodes along the path $0, 1, \ldots, D$. For each requested job c, scheduler $\mathcal{A}$ requests a set S'_c of v_c/d_c calls for $\mathcal{A}'$, where the ith call starts at node $[(i-1) \cdot \frac{Dd_c}{v_c}]$ and ends at node $[i \cdot \frac{Dd_c}{v_c}]$. (That is, $N_c = v_c/d_c$ and each generated call c' is of length $r_{c'} = D\frac{d_c}{v_c}$. All these calls can be served "back to back" on the path.) All generated calls have $l_{c'} = 1$. Next, $\mathcal{A}$ maintains the property that call c is in service with probability proportional to the fraction of the corresponding c' calls currently served by $\mathcal{A}'$.

The expected gain of $\mathcal{A}'$ from the jobs in S'_c is $d_c n_c(e_c)$, and the expected gain of $\mathcal{A}$ from job c is $v_c n_c(e_c)/N_c = d_c n_c(e_c)$. Thus, the expected gain of $\mathcal{A}$ on its input sequence S is equal to the expected gain of $\mathcal{A}'$ on the constructed sequence S'. Since $\mu_d(S) = \mu_d(S')$ and $\mu_v(S) \cdot \mu_d(S) \geq D$, the existence of $\mathcal{A}'$ contradicts Theorem 2.

(ii) As in the proof of (i), $\mathcal{A}$ picks a path of length D in the tree. The vertices of this path are labeled $\{0, \ldots, D\}$. $N_c = 1$, $l_c = 1$, $d_c = d_{c'}$. The endpoints of the call in S'_c are node 0 and node v_c. Thus, $v_c = r_{c'} = v_{c'}$. Since $\mu_d(S) = \mu_d(S')$ and $\mu_v(S) = D$, the existence of $\mathcal{A}'$ contradicts Theorem 2.

(iii) The construction for (ii) works here since for all c, $l_c = 1$.

(iv) As in the proof of (i), $\mathcal{A}$ picks a path of length D in the tree. The vertices of this path are labeled $\{0, \ldots, D\}$. $N_c = 1$, $l_c = 1$, $d_c = d_{c'}$. The endpoints of the call in S'_c are 0 and node v_c/d_c. Thus, $r_{c'} = v_c/d_c$ and $v_c = r_c d_{c'} = v_{c'}$. Since $\mu_d(S) = \mu_d(S')$ and $\mu_v(S) \cdot \mu_d(S) \geq D$, the existence of $\mathcal{A}'$ contradicts Theorem 2.

(v) The construction for (iv) works here since for all c, $l_c = 1$. ∎

5. Tightness of the bound. We sketch two simple preemptive, randomized schedulers in the basic model defined in section 2. The first is $O(\log \mu_v)$-competitive and the second is $O(\log \mu_d)$-competitive. Depending on whether $\mu_d \leq \mu_v$, we can implement the appropriate one to obtain an $O(\log \mu)$-competitive scheduler for $\mu = \min\{\mu_v, \mu_d\}$. In both cases we use a technique of [3].

For the $O(\log \mu_v)$-competitive scheduler, let m (resp., M) be the minimum (resp., maximum) possible value of a job. Let $k = \lceil \log(M/m) \rceil$ and let $v_0, \ldots, v_k$ satisfy $v_0 = m$, $v_k = M$, and $v_i/v_{i-1} \leq 2$. The scheduler first picks $1 \leq i \leq k$ at random. Next, it rejects all jobs whose value does not fall between v_{i-1} and v_i. Jobs with value between v_{i-1} and v_i are scheduled as follows: accept an incoming job if it terminates before the currently running job. This scheduler is 2-competitive over the sequence of jobs in the chosen range. Thus, the competitive ratio of the randomized algorithm is $4 \log_2 \mu_v = 4 \log_2(M/m)$.

For the $O(\log \mu_d)$-competitive scheduler, let d (resp., D) be the minimum (resp., maximum) possible duration of a job. Let $k = \lceil \log(D/d) \rceil$ and let $d_0, \ldots, d_k$ satisfy $d_0 = m$, $d_k = M$, and $d_i/d_{i-1} \leq 2$. The scheduler first picks $1 \leq i \leq k$ at random. Next, it rejects all jobs whose duration does not fall between d_{i-1} and d_i. Jobs with duration between d_{i-1} and d_i are scheduled as follows: accept an incoming job if there is no currently running job or if the new job has at least twice the value of the currently running job.

We first show that the competitive ratio of this scheduler is 14-competitive, assuming that all jobs have the same duration. Then we employ the above [3] technique to argue that the competitive ratio of this scheduler is $O(\log_2 \mu_d) = O(\log_2(M/m))$.

Assume all jobs have the same duration. First we observe that the total value of all jobs which the scheduler preempts is at most the total value of the jobs which the scheduler completes. This follows from the fact that the scheduler only preempts a job in favor of a job which is at least twice the value of the currently running job. Next we bound the gain from jobs which the scheduler rejects (i.e., never schedules). Each rejected job intersects in time with a running job. (The running job can later be either completed or rejected.) It can be seen that the maximum gain from rejected jobs that intersect each running job c is at most $6v_c$. Thus the maximum gain from rejected jobs is at most 6 times the total gain from completed or preempted jobs. Altogether, the maximum gain from rejected, preempted, and completed jobs is at most 14 times the gain from completed jobs.

In the above construction it is assumed that the scheduler would know μ_d and μ_v in advance. We remark that this can be avoided by "rerandomizing" every time a job is introduced which increases μ_d or μ_v. We do not elaborate here.

6. Open questions. The first question we leave open is to close the quadratic gap for our generic scheduling problem. A logarithmic lower bound would not only match the upper bound shown in section 5 but would also match the other logarithmic upper bounds discussed in section 1.1.

Although much of the recent work in online routing in communication networks addresses the limited capacity problem, most of the work on the more traditional load balancing problems relates to makespan minimization. Despite the general lack of attention, the limited capacity problem seems to be well suited to many of the applications for the variety of scheduling paradigms in the literature. This points out an important and largely unexplored area in online scheduling. For example, there has been little which addresses admission control problems related to either example (a) or (b) at the beginning of the introduction. Although the work of [10], [11], and

[22] is an important step in this direction, that work addresses a very special case. Thus, generalizing these results to apply to more complex models is an important and relevant new direction for research.

Acknowledgments. We are grateful to the hospitality of Zvi Galil and Columbia University that enabled us to carry out this work during the summer of 1994. We also thank the anonymous referees for their very helpful comments.

REFERENCES

[1] J. Aspens, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts, *On-line load balancing with applications to machine scheduling and virtual circuit routing*, in Proc. 25th ACM Symposium on the Theory of Computing, 1993, pp. 623–631.

[2] B. Awerbuch, Y. Azar, and S. Plotkin, *Throughput-competitive online routing*, in 34th IEEE Symposium on Foundations of Computer Science, 1993, pp. 32–40.

[3] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosen, *Competitive non-preemptive call control*, in Proc. 5th ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, PA, 1994, pp. 312–320.

[4] B. Awerbuch, R. Gawlick, T. Leighton, and Y. Rabani, *On-line admission control and circuit routing for high performance computing and communication*, in Proc. 35th Annual IEEE Symposium on the Foundations of Computer Science, 1994, pp. 412–423.

[5] Y. Azar, A. Z. Broder, A. R. Karlin, *Online load balancing*, in Proc. 33rd Annual IEEE Symposium on the Foundations of Computer Science, 1992, pp. 218–225.

[6] Y. Azar, B. Kalyanasundaram, S. Plotkin, K. Pruhs, and O. Waarts, *Online load balancing of temporary tasks*, in Workshop on Algorithms and Data Structures, 1993, pp. 119–130; J. Algorithms, 22 (1996), pp. 93–110.

[7] Y. Azar, J. Naor, and R. Rom, *The competitiveness of on-line assignments*, in Proc. 3rd ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, PA, 1992, pp. 203–210.

[8] A. Bar-Noy, R. Canetti, S. Kutten, Y. Mansour, and B. Schieber, *Bandwidth allocation with preemption*, in Proc. 27th ACM Symposium on Theory of Computing, 1995, pp. 616–625.

[9] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra, *New algorithms for an ancient scheduling problem*, in Proc. 24th ACM Symposium on Theory of Algorithms, 1992, pp. 51–58; J. Comput. System Sci., 51 (1995), pp. 359–366.

[10] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha, *On-line scheduling in the presence of overload*, in Proc. 32nd Annual IEEE Symposium on the Foundations of Computer Science, 1991, pp. 100–110.

[11] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang, *On the competitiveness of on-line real-time task scheduling*, J. Real-Time Systems, 4 (1992), pp. 124–144.

[12] J. A. Garay and I. S. Gopal, *Call preemption in communication networks*, in Proceedings of INFOCOM '92, IEEE, 1992.

[13] J. Garay, I. S. Gopal, S. Kutten, Y. Mansour, and M. Yung, *Efficient on-line call control algorithms*, in Proc. 2nd Israel Symposium on Theory of Computing and Systems, 1993, pp. 285–293; J. Algorithms, 23 (1997), pp. 180–194.

[14] R. L. Graham, *Bounds on multiprocessing timing anomalies*, SIAM J. Appl. Math., 17 (1969), pp. 416–429.

[15] M. Harchol-Balter and A. B. Downey, *Exploiting Process Lifetime Distributions for Dynamic Load Balancing*, Tech. Report TR-95-021, International Computer Science Institute, Berkeley, CA, 1995.

[16] D. R. Karger, S. J. Phillips, and E. Torng, *A better algorithm for an ancient scheduling problem*, J. Algorithms, 20 (1996), pp. 400–430.

[17] G. Koren and D. Shasha, *D^{over}: An Optimal On-Line Scheduling Algorithm for Overloaded Real-Time Systems*, Tech. Report 594, Courant Institute, New York University, 1992.

[18] G. Koren and D. Shasha, *MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling*, Theoret. Comput. Sci., Special Issue on Dependable Parallel Computing, 128 (1994), pp. 75–97.

[19] R. Motwani, S. Phillips, and E. Torng, *Non-clairvoyant scheduling*, Theoret. Comput. Sci., 130 (1994), pp. 17–47.

[20] S. Phillips and J. Westbrook, *On-line load balancing and network flow*, in Proc. 25th ACM Symposium on Theory of Computing, 1993, pp. 402–411.

[21] K. K. Ramakrishnan, L. Vaitzblit, C. Gray, U. Vahalia, D Ting, P. Tzelnic, S. Glaser, and W. Duso, *Operating system support for a video-on-demand file service*, in Network and Operating System Support for Digital Audio and Video: Fourth International Workshop, Springer-Verlag, Lancaster, UK, 1993, pp. 225–236.

[22] G. Woeginger, *On-line scheduling of jobs with fixed start and end times*, Theoret. Comput. Sci., 130 (1994), pp. 5–16.

[23] *Special Issue on Asynchronous Transfer Mode*, Internat. J. Digital Analog Cabled Systems, 1 (4), 1988.

SIAM J. COMPUT.
Vol. 27, No. 4, pp. 1016–1035, August 1998
© 1998 Society for Industrial and Applied Mathematics
006

SURFACE APPROXIMATION AND GEOMETRIC PARTITIONS*

PANKAJ K. AGARWAL† AND SUBHASH SURI‡

Abstract. Motivated by applications in computer graphics, visualization, and scientific computation, we study the computational complexity of the following problem: given a set S of n points sampled from a bivariate function $f(x, y)$ and an input parameter $\varepsilon > 0$, compute a piecewise-linear function $\Sigma(x, y)$ of minimum complexity (that is, an xy-monotone polyhedral surface, with a minimum number of vertices, edges, or faces) such that $|\Sigma(x_p, y_p) - z_p| \leq \varepsilon$ for all $(x_p, y_p, z_p) \in S$. We give hardness evidence for this problem, by showing that a closely related problem is *NP-hard*. The main result of our paper is a polynomial-time approximation algorithm that computes a piecewise-linear surface of size $O(K_o \log K_o)$, where K_o is the complexity of an optimal surface satisfying the constraints of the problem.

The technique developed in our paper is more general and applies to several other problems that deal with partitioning of points (or other objects) subject to certain geometric constraints. For instance, we get the same approximation bound for the following problem arising in machine learning: given n "red" and m "blue" points in the plane, find a minimum number of *pairwise disjoint* triangles such that each blue point is covered by some triangle and no red point lies in any of the triangles.

Key words. approximation algorithms, dynamic programming, levels of detail, machine learning, terrains, simplification, visualization

AMS subject classifications. 65Y25, 68Q20, 68Q25, 68U05.

PII. S0097539794269801

1. Introduction.

In scientific computation, visualization, and computer graphics, the modeling and construction of surfaces is an important area. A small sample of some recent papers [1, 2, 4, 8, 11, 14, 22, 23] on this topic gives an indication of the scope and importance of this problem. Rather than delve into any specific problem studied in these papers, we focus on a general, abstract problem that seems to underlie them all.

In many scientific and computer graphics applications, computation takes place over a surface in three dimensions. The surface is generally modeled by piecewise linear (or sometimes piecewise cubic) patches, whose vertices lie either on or in the close vicinity of the actual surface. In order to ensure that all local features of the surface are captured, algorithms for an automatic generation of these models generally sample at a dense set of regularly spaced points. Demands for real-time speed and reasonable performance, however, require the models to have as small a combinatorial complexity as possible. A common technique employed to reduce the complexity of the model is to somehow "thin" the surface by deleting vertices with relatively "flat" neighborhoods. Only ad hoc and heuristic methods are known for this key step. Most of the thinning methods follow a set of local rules (such as deleting edges or vertices whose incident faces are almost coplanar), which are applied in an arbitrary order

*Received by the editors June 10, 1994; accepted for publication (in revised form) May 22, 1996; published electronically May 19, 1998.

http://www.siam.org/journals/sicomp/27-4/26980

†Department of Computer Science, Box 1029, Duke University, Durham, NC 27708-0129 (pankaj@euclid.cs.duke.edu). The work of this author was supported by National Science Foundation research grant CCR-93-01259, by Army Research Office MURI grant DAAH04-96-1-0013, by a Sloan fellowship, by a National Science Foundation NYI award and matching funds from Xerox Corp., and by a grant from the U.S.-Israeli Binational Science Foundation.

‡Department of Computer Science, Washington University, Campus Box 1045, One Brookings Drive, St Louis, MO 63130 (suri@cs.wustl.edu).

until they are no longer applicable. Not surprisingly, these methods come with no performance guarantee, and generally no quantitative statement can be made about the surface approximation computed by them.

In this paper, we address the complexity issues of the surface approximation problem for surfaces that are xy-monotone. These surfaces represent the graphs of continuous bivariate functions $f(x, y)$, and they arise quite naturally in many scientific applications. One possible approach for handling arbitrary surfaces is to break them into monotone pieces and apply our algorithm individually on each piece. Let us formally define the main problem studied in our paper.

Let f be a continuous function of two variables x and y, and let S be a set of n points sampled from f. A continuous piecewise-linear function Σ is called an *ε-approximation* of f, for $\varepsilon > 0$, if

$$|\Sigma(x_p, y_p) - z_p| \leq \varepsilon$$

for every point $p = (x_p, y_p, z_p) \in S$. Given S and ε, the *surface-approximation* problem is to compute a piecewise-linear function Σ that ε-approximates f with a minimum number of breakpoints. The breakpoints of Σ can be arbitrary points of $\mathbb{R}^3$, and they are not necessarily points of S. In many applications, f is generally a function hypothesized to fit the observed data—the function Σ is a computationally efficient substitute for f. The parameter ε is used to achieve a complexity–quality tradeoff: the smaller the ε, the higher the fidelity of the approximation. (The graph of a piecewise-linear function of two variables is also called a *polyhedral terrain* in computational geometry literature; the breakpoints of the function are the vertices of the terrain.)

Although there is a vast literature on the surface-approximation problem in the graphics community, the state of theoretical knowledge of this problem appears to be rather slim. The provable performance bounds are known only for convex surfaces. For this special case, an $O(n^3)$ time algorithm is presented by Mitchell and Suri [17] for computing an ε-approximation of a convex polytope of n vertices in $\mathbb{R}^3$. Their algorithm produces an approximation of size $O(K_o \log n)$, where K_o is the size of an optimal ε-approximation. Extending their work, Clarkson [7] has proposed an $O(K_o n^{1+\delta})$ expected time randomized algorithm for computing an approximation of size $O(K_o \log K_o)$, where δ can be an arbitrarily small positive number. Recently Brönnimann and Goodrich [5] have refined Clarkson's algorithm and have given a polynomial-time algorithm for computing an approximation of size $O(K_0)$.

In this paper, we study the approximation problem for surfaces that correspond to graphs of bivariate functions. We give evidence that the surface approximation may be hard, by showing that a closely related problem is *NP-hard*. In particular, we prove that it is *NP-hard* to decide whether the set of sample points can be ε-approximated with k triangles whose projections on the $z = 0$ plane are pairwise disjoint. (The latter problem relaxes the *continuity* requirement of the surface approximation.)

The main result of our paper, however, is a polynomial-time algorithm for computing an ε-approximating surface of a guaranteed quality. If an optimal ε-approximating surface of f has K_o vertices, then our algorithm produces a surface with $O(K_o \log K_o)$ vertices. (The number of vertices, edges, and faces are linearly related by Euler's formula, and so the approximation holds for any of the three measures.) Observe that we are dealing with two approximation measures here: ε, which measures the absolute z difference between f and the "simplified" surface Σ, and the ratio between the sizes of the optimal surface and the output of our algorithm. For lack of better terminology,

we use the term "approximation" for both measures. Notice, though, that ε is an input (user-specified) parameter, and $\log K_o$ is the approximation guarantee provided by the analysis of our algorithm.

The key to our approximation method is an algorithm for partitioning a set of points in the plane by *pairwise disjoint* triangles. This is an instance of the geometric set-cover problem, with an additional *disjointness* constraint on the covering objects (triangles). Observe that the disjointness condition on covering objects precludes the well-known *greedy* method for set covering [12, 15]; in fact, we can show that a greedy solution has size $\Omega({K_o}^2)$ in the worst case. Let us now reformulate our surface-approximation problem as a constrained geometric partitioning problem.

Let $\bar{p}$ denote the orthogonal projection of a point $p \in \mathbb{R}^3$ onto the xy-plane $z = 0$. In general, for any set $A \subset \mathbb{R}^3$, we use $\bar{A}$ to denote the orthogonal projection of A onto the xy-plane. Then, in order to get an ε-approximation of f, it suffices to find a set of triangles in 3-space so that (i) the projections of these triangles on plane $z = 0$ are pairwise disjoint and they cover the projected set of points $\bar{S}$, and (ii) the vertical distance between a triangle and any point of S directly above/below it is at most ε. Our polynomial-time algorithm produces a family of $O(K_o \log K_o)$ such triangles. We stitch together these triangles to produce the desired surface Σ. The "stitching" process introduces at most a constant factor more triangles.

The geometric partition framework also includes several extensions and generalizations of the basic surface-approximation problem. For instance, we can formulate a stronger version of the problem by replacing each sample point by a horizontal triangle (or any polygon). Specifically, we are given a family of *horizontal* triangles (or polygons) in 3-space, whose projections on the xy-plane are pairwise disjoint. We want a piecewise-linear, ε-approximating surface whose maximum vertical distance from *any* point on the triangles is ε. Our approximation algorithm works equally well for this variant of the problem—this variant addresses the case when some local features of the surface are known in detail; unfortunately, our method works only for horizontal features.

Finally, let us mention the *planar bichromatic partition* problem, which is of independent interest in the machine learning literature. Given a set R of "red" points and another set B of "blue" points in the plane, find a minimum number of pairwise disjoint triangles so that each blue point lies in a triangle and no red point lies in any of the triangles. Our algorithm gives a solution with $O(K_o \log K_o)$ triangles. It can also be used to construct a linear decision tree of size $O(K_o \log K_o)$, which is consistent with respect to R and B, where K_o is the size of an optimal linear decision tree.

The running time of our algorithms, though polynomial, is quite high, and at the moment has only theoretical value. These being some of the first results in this area, however, we expect that the theoretical time complexity of these problems would improve with further work. Perhaps some of the ideas in our paper may also shed light on the theoretical performance of some of the "practical" algorithms that are used in the trade.

2. A proof of NP-hardness. We first show that the planar bichromatic partition problem is *NP-hard*, by a reduction from the planar 3-SAT. Later we extend the proof to the surface approximation by pairwise disjoint triangles. We do not know whether they are in *NP* since the coordinates of the triangles in the solution may require very high precision. We recall that the 3-SAT problem consists of n variables $x_1, \ldots, x_n$, and m clauses $C_1, \ldots, C_m$, each with three literals C_{i1}, C_{i2}, C_{i3}, where

C_{ij} is either x_k or $\bar{x}_k$. The problem is to decide whether the boolean formula

$$F = \bigwedge_{i=1}^{m} (C_{i1} \vee C_{i2} \vee C_{i3})$$

has a truth assignment. An instance of 3-SAT is called *planar* if its variable-clause graph is planar. In other words, F is an instance of the planar 3-SAT if the graph $G(F) = (V, E)$ is planar (see [13]), where V and E are defined as follows:

$$V = \{x_1, x_2, \ldots, x_n\},$$
$$E = \{(x_j, C_i) \mid x_i \text{ or } \bar{x}_i \text{ appears in } C_i\}.$$

THEOREM 2.1 (planar bichromatic partition problem). *The following problem is* NP-hard. *Given a set R of "red" points, another set B of "blue" points in the plane, and an integer parameter k, do there exist k pairwise disjoint triangles containing all the blue points and none of the red points?*

Proof. Our construction is similar to the one used by Fowler, Paterson, and Tanimoto [10], who prove the intractability of certain planar geometric covering problems (without the disjointness condition); see also [3, 9] for similar constructions. We first describe our construction for the bichromatic partition problem. To simplify the proof, our construction allows three or more points to lie on a line—the construction can be modified easily to remove these degeneracies.

Let F be a boolean formula, and let $G = (V, E)$ be a straight-line planar embedding of the graph $G(F)$. We construct an instance of the bichromatic partition problem whose solution determines whether F is satisfiable.

We start by placing a "blue" point at each clause node C_j, $1 \le j \le m$. Let x_i be a variable node, and let $e_{i1}, e_{i2}, \ldots, e_{il}$ be the edges incident to it. In the plane embedding of G, the edges e_{ij} form a "star" (see Figure 1 (i)). We replace this star by its Minkowski sum with a disk of radius δ, for a sufficiently small $\delta > 0$. Before performing the Minkowski sum, we shrink each edge e_{ij} by 2δ at its endpoint C_{ij}, so that different "star-shaped polygons" meeting at a clause node do not overlap (see Figure 1 (ii)). Let P_i denote the star-shaped polygon corresponding to x_i. In the polygon P_i, there is a tube corresponding to each edge e_{ij}. The tube consists of two copies of e_{ij}, each translated by distance δ, plus a circular arc s_{ij} near the clause node C_j.

We place an even number of (say, $2k_i$) "blue" points on the boundary of P_i, as follows. We put two points a_{ij} and b_{ij} on the circular arc s_{ij} near its tip. If C_j contains the literal x_i, we put six points on the straight-line portion of P_i's boundary, three each on translated copies of the edge e_{ij}. On each copy, we move the middle point slightly inwards so as to replace the original edge of P_i by a path of length 2. On the other hand, if C_j contains the literal $\bar{x}_i$, we put four points on the straight-line portion of P_i's boundary, two each on translated copies of the edge e_{ij}. Thus, the number of blue points added on the tube corresponding to the edge e_{ij} is either six or eight. ($2k_i$ is the total number of points put along P_i.) Let B denote the set of all blue points placed in this way, and let $k = \sum_{i=1}^{n} k_i$.

Finally, we scatter a large (but polynomially bounded) number of "red" points so that

(i) any segment connecting two blue points that are not adjacent along the boundary of some P_i contains a red point, and

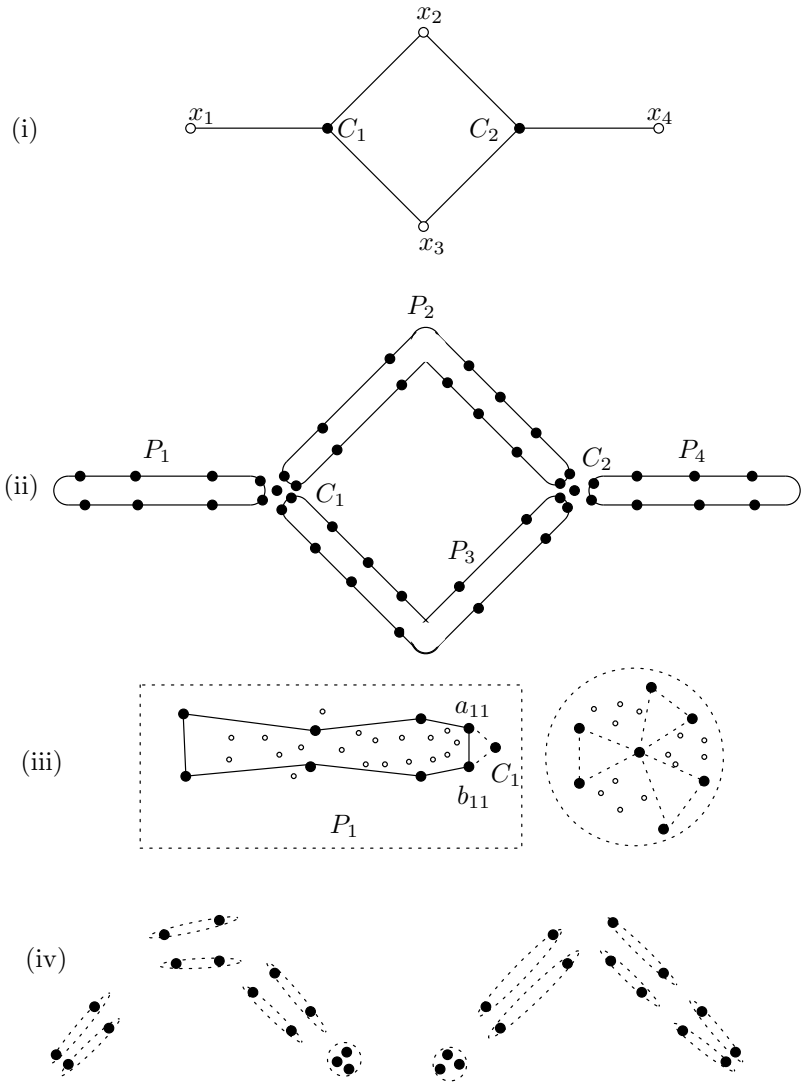


FIG. 1. (i) *An instance of planar* 3*-SAT:* $F = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_2 \vee \bar{x}_3 \wedge x_4)$. (ii) *Corresponding instance of bichromatic partition.* (iii) *Details of* P_1 *and* C_1*; only some of the red points lying near* P_1 *and* C_1 *are shown.* (iv) *Two possible coverings of blue points on* P_2.

(ii) any triangle with three blue points as its vertices contains at least one red point unless the triangle is defined by $a_{ij}b_{ij}C_j$ for some edge $e_{ij} \in E$. (See Figure 1 (iii).)

Let R be a set of red points satisfying the above two properties.

We claim that the set of blue points B can be covered by k pairwise disjoint triangles, none of which contains any red point, if and only if the formula F has a truth assignment. Our proof is similar to the one in Fowler, Paterson, and Tanimoto [10]; we only sketch the main ideas. The red points act as enforcers, ensuring that only those blue points that are adjacent on the boundary of a P_i can be covered by a single triangle. Thus, the minimum number of triangles needed to cover all the points on P_i

is k_i. Further, there are precisely two ways to cover these points using k_i triangles: in one covering, a_{ij} and b_{ij} are covered by a single triangle for those clauses only in which $x_i \in C_j$, and, in the other covering, a_{ij} and b_{ij} are covered by a single triangle for those clauses only in which $\bar{x}_i \in C_j$; see Figure 4 (iv). We regard the first covering as setting $x_i = 1$, and the second covering as setting $x_i = 0$.

Suppose $x_i = 1$. For any clause C_j that contains x_i, the points a_{ij} and b_{ij} are covered by a single triangle, and we can cover the clause point corresponding to C_j by the same triangle. The same holds if $x_i = 0$ and the clause C_j contains $\bar{x}_i$. In other words, the clause point of C_j can be covered for free if C_j is satisfied. Thus, the set of blue points B can be covered by k triangles if and only if the clause point for each clause C_j is covered for free, that is, the formula F has a truth assignment. This completes our proof of *NP*-hardness of the planar bichromatic partition problem. □

REMARK. The preceding construction is degenerate in that most of the red points lie on segments connecting two blue points. There are several ways to remove these collinearities; we briefly describe one of them. For each polygon P_i, replace every blue point b on P_i by two blue points b', b'', placed very close to b. (We do not make copies of "clause points" C_j, $1 \le j \le m$.) For every pair of blue points b_j, b_l that we did not want to cover by a single triangle in the original construction, we place a red point in the convex hull of b'_j, b''_j, b'_l, b''_l. If there are $4k_i$ blue points on the boundary of P_i, they can be covered by k_i triangles, and there are exactly two ways to cover these blue points by k_i triangles, as earlier. Following a similar, but more involved, argument, we can prove that the set of all blue points can be covered by $\sum_{i=1}^{n} k_i$ triangles if and only if F is satisfiable.

THEOREM 2.2. *The following problem is* NP-hard. *Given a set S of n points sampled from a bivariate function $f(x, y)$, a real parameter $\varepsilon > 0$, and an integer k, can the points of S be ε-approximated by k triangles whose projections on the plane $z = 0$ are pairwise disjoint?*

Proof. Our construction is similar in spirit to the one for the bichromatic partition problem, albeit slightly more complex in detail. We use points of three colors: red, white, and black. The "white" points lie on the plane $z = 0$, the "black" points lie on the plane $z = 2A$, and the "red" points lie between $z = 0$ and $z = A$, where A is a sufficiently large constant. To maintain a connection with the previous construction, the black and white points play the role of blue points, while the red points play the role of enforcers as before, restricting the choice of "legal" triangles that can cover the black or white points. We will describe the construction in the xy-plane, which represents the orthogonal projection of the actual construction. The actual construction is obtained simply by vertically translating each point to its correct plane, depending on its color.

We start out again by putting a "black" point at each clause node C_j. Then, for each variable x_i, we construct the "star-shaped" polygon P_i; this part is identical to the previous construction. We replace each of the two straight-line edges of P_i by "concave chains," bent inward, and also make a small "dent" at the tip of the circular arc s_{ij}, as shown in Figure 2. We place 12 points on each arm of P_i, alternating black and white, as follows. At the tip of the circular arc s_{ij}, we put a white point c_{ij} at the outer endpoint of the dent and a black point d_{ij} at the inner endpoint of the dent (Figure 2 (ii)). The rest of the construction is shown in Figure 2 (i): we put two more points a_{ij}, b_{ij} on the circular arc and four points $\alpha_{ij}^l, \beta_{ij}^l$ $(1 \le l \le 4)$ on each of the two concave chains. The two points surrounding c_{ij}, d_{ij}, namely, a_{ij} and b_{ij}, are such that any segment connecting them to any point on the two concave chains lies inside

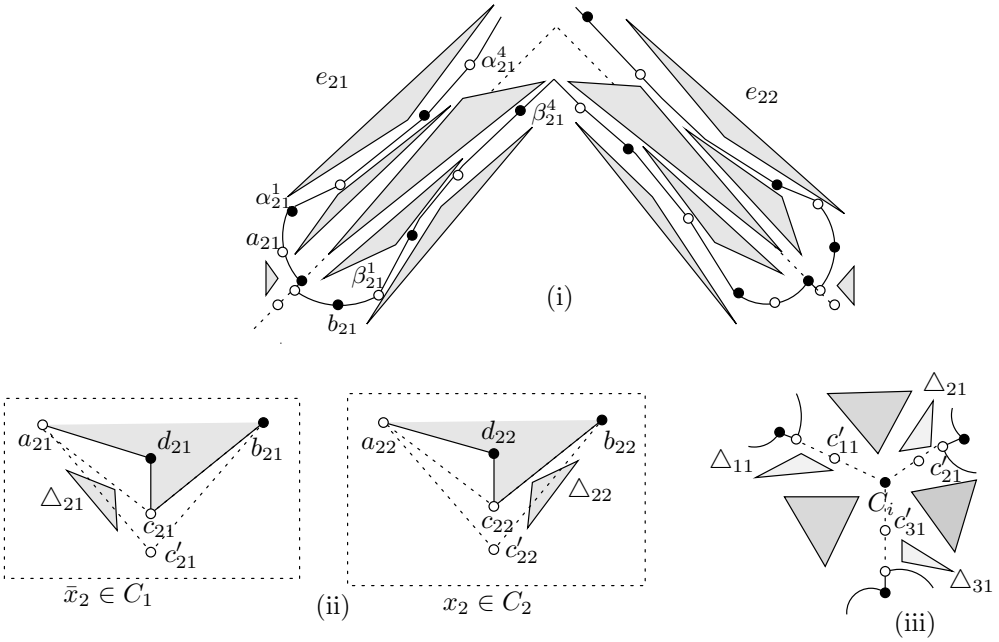


FIG. 2. *Placing points on the polygon P_2 corresponding to Figure* 1. (i) *Modified P_2 and points on P_2.* (ii) *Points and triangles in the neighborhood of c_{21}, c_{22}.* (iii) *Points and triangles near C_1.*

P_i. Next, corresponding to each edge e_{ij} of the graph $G(F)$, we put a "white" point c'_{ij} on the segment joining c_{ij} and the clause point C_j, very close to c_{ij}, such that

$$d(c_{ij}, c'_{ij}) \leq \frac{\varepsilon}{2A} d(c_{ij}, C_j)\,.$$

(See Figure 2 (iii).) This condition says that, in the final construction when the black and white points have been translated to their correct z-plane, the vertical distance between c'_{ij} and the segment $C_j c_{ij}$ is no more than ε—recall that ε is the input measure of approximation.

This completes the placement of white and black points. The only remaining part of the construction is the placement of "red" points, which we now describe.

We add a set of triangles, each containing a large (but polynomially bounded) number of red points—the role of these triangles is to restrict the choice of legal triangles that can cover black/white points. The set of triangles associated with P_i is labeled T_i. The construction of T_i is detailed in Figure 2 (i). Specifically, for an edge e_{ij}, if $x_i \in C_j$, then we add a small triangle that intersects the segment $b_{ij}c'_{ij}$ but not $b_{ij}c_{ij}$. On the other hand, if $\bar{x}_i \in C_j$, then we add a small triangle that intersects $a_{ij}c'_{ij}$ but not $a_{ij}c_{ij}$. Next, we add a small number of triangles inside P_i, near its concave chains, so that at most three consecutive points along P_i may be covered by one triangle without intersecting any triangle of T_i. We ensure that one of these triangles intersects the triangle $\triangle\alpha^1_{ij}a_{ij}d_{ij}$, so that $\{\alpha^1_{ij}, a_{ij}, d_{ij}\}$ cannot be covered by a single triangle. We also place three triangles near each clause C_j, each containing a large number of red points; see Figure 2 (iii). Finally, we translate black and white points in the z-direction, as described earlier. Let $\{\tau_1, \ldots, \tau_t\}$ be the set of all "red" triangles. We move all points in τ_i vertically to the plane $z = \frac{A}{4}(1 + \frac{2i}{t})$.

Now, suppose that polygon P_i has $3k_i$ white or black points on its boundary.

Then, there are two ways to cover the points of P_i with k_i legal (nonintersecting) triangles: one in which a_{ij}, d_{ij}, c_{ij} are covered by a single triangle, and one in which b_{ij}, d_{ij}, c_{ij} are covered by a triangle. These coverings are associated with the true and false settings of the variable x_i. Let P denote the set of all points constructed, and let $k = \sum_{i=1}^{n} k_i$. Let t denote the total number of "red" triangles. Recall also that m is the number of clauses in F.

We claim that the set of points P can be ε-approximated with $k+m+t$ triangles if and only if F has a truth assignment, provided that ε is sufficiently small. The claim follows from the observations that it is always better to cover all red points lying in a horizontal triangle τ_i by τ_i itself, and that a clause C_j requires one triangle to cover its points if and only if one of the literals in C_j is set true; otherwise it requires two triangles. (For instance, if C_j contains the literal x_i and x_i is set true, then the triangle a_{ij}, d_{ij}, c_{ij} can be enlarged slightly to cover c'_{ij}. The remaining three points for the clause C_j can be covered by one additional triangle.) The rest of the argument is the same as for the bichromatic partition problem. Finally, we can perturb the points slightly so that no four of them are coplanar. □

REMARK. Our proof has some technical problems if the approximating surface is required to be continuous. While the surface approximation and the approximation by disjoint triangles seem closely related, we can only claim to have proved the hardness for the latter. In the remainder of the paper, however, we develop algorithms for the continuous version of the surface approximation; of course, our algorithmic results hold for the problem of approximation by triangles as well.

3. A canonical trapezoidal partition. We introduce an abstract geometric partitioning problem in the plane, which captures the essence of both the surface-approximation problem and the bichromatic partition problem. The abstract problem deals with trapezoidal partitions under a boolean constraint function $\mathcal{C}$ satisfying the "subset restriction" property. More precisely, let $\mathcal{C}$ be a boolean function from compact, connected subsets of the plane to $\{0, 1\}$ satisfying the following property:

$$\mathcal{C}(U) = 1 \implies \mathcal{C}(V) = 1 \qquad \text{for all} \quad V \subseteq U \subseteq \mathbb{R}^2. \tag{3.1}$$

For technical reasons, we choose to work with "trapezoids" instead of triangles, where the top and bottom edges of each trapezoid are parallel to the x-axis. The trapezoids and triangles are equivalent for the purpose of approximation: each triangle can be decomposed into two trapezoids, and each trapezoid can be decomposed into two triangles.

Given a set of n points P in the plane, a family of trapezoids $\mathbf{\Delta} = \{\Delta_1, \ldots, \Delta_m\}$ is called a *valid trapezoidal partition* (a *trapezoidal partition*, for brevity) of P with respect to a boolean constraint function $\mathcal{C}$ if the following conditions hold:

(i) $\mathcal{C}(\Delta) = 1$, for all $\Delta \in \mathbf{\Delta}$;
(ii) $\mathbf{\Delta}$ covers all the points: $P \subset \bigcup_{i=1}^{m} \Delta_i$; and
(iii) the trapezoids in $\mathbf{\Delta}$ have pairwise disjoint interiors.

We can cast our bichromatic partition problem in this abstract framework by setting $P = B$ (the set of "blue" points) and, for a trapezoid $\tau \subset \mathbb{R}^2$, defining $\mathcal{C}(\tau) = 1$ if and only if τ is empty of red points, that is, $\tau \cap R = \emptyset$. In the surface-approximation problem, we set $P = \bar{S}$ (the orthogonal projection of S on the plane $z = 0$), and a trapezoid $\tau \subset \mathbb{R}^2$ has $\mathcal{C}(\tau) = 1$ if and only if τ can be vertically lifted to a planar trapezoid $\hat{\tau}$ in $\mathbb{R}^3$ so that the vertical distance between $\hat{\tau}$ and any point of S directly above/below it is at most ε.

The space of optimal solutions for our abstract problem is potentially infinite—the vertices of the triangles in our problem can be anywhere in the plane. For our approximation results, however, we show that a restricted choice of trapezoids suffices.

Given a set of n points P in the plane, let $\mathcal{L}(P)$ denote the set consisting of the following lines: the horizontal lines passing through a point of P, and the lines passing through two points of P. Thus, $|\mathcal{L}(P)| = O(n^2)$. We will call the lines of $\mathcal{L}(P)$ the *canonical lines* determined by P. We say that a trapezoid $\Delta \subset \mathbb{R}^2$ is *canonical* if all of its edges belong to lines in $\mathcal{L}(P)$. A trapezoidal partition $\mathbf{\Delta}$ is *canonical* if all of its trapezoids are canonical (see Figure 3). The following lemma shows that by limiting ourselves to canonical trapezoidal partitions only, we sacrifice at most a constant (multiplicative) factor in our approximation.

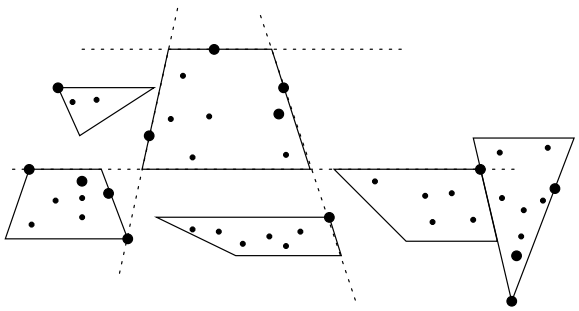


FIG. 3. *A canonical trapezoidal partition.*

LEMMA 3.1. *Any trapezoidal partition of P with k trapezoids can be transformed into a canonical trapezoidal partition of P with at most $4k$ trapezoids.*

Proof. We give a construction for transforming each trapezoid $\Delta \in \mathbf{\Delta}$ into four trapezoids $\Delta_i \subseteq \Delta$, for $1 \leq i \leq 4$, with pairwise disjoint interiors, so that Δ_i together cover all the points in $P \cap \Delta$. By (3.1), the new set of trapezoids is a valid trapezoidal partition of P. Our construction works as follows.

Consider the convex hull of the points $P_\Delta = P \cap \Delta$. If the convex hull itself is a trapezoid, we return that trapezoid. Otherwise, let ℓ, r, t, b denote the left, right, top, and bottom edges of Δ, as shown in Figure 4 (i). We perform the following four steps, which constitute our transformation algorithm.

(i) We shrink the trapezoid Δ by translating each of its four bounding edges towards the interior, until it meets a point of P_Δ. Let $\Delta' \subseteq \Delta$ denote the smaller trapezoid thus obtained (Figure 4 (i)). Let p_ℓ, p_r, p_t, p_b, respectively, denote a point of P_Δ lying on the left, right, top, and bottom edges of Δ'; we break ties arbitrarily if more than one point lies on an edge.

(ii) We partition Δ' into two trapezoids, Δ_L and Δ_R, by drawing the line segment $p_u p_b$, as shown in Figure 4 (ii).

(iii) We next partition Δ_L into two trapezoids Δ_{LU} and Δ_{LB}, by drawing the maximal horizontal segment through p_ℓ. Let p'_ℓ denote the right endpoint of this segment. Similarly, we partition Δ_R into Δ_{RU} and Δ_{RB}, lying, respectively, above and below the horizontal line segment $p_r p'_r$.

(iv) We rotate the line supporting the left boundary of Δ_{LU} around the point p_ℓ in clockwise direction until it meets a point of the set $P \cap \Delta_{LU}$. Let q_ℓ denote the intersection of this line and the top edge of Δ_{LU}. We set $\Delta_1 = p_\ell q_\ell p_u p'_\ell$ (Figure 4 (iv)). (If $q_\ell = p_u$, then Δ_1 is a triangle, which we regard as a degenerate trapezoid; e.g., Δ_4 in Figure 4 (iii).) The top and bottom edges of

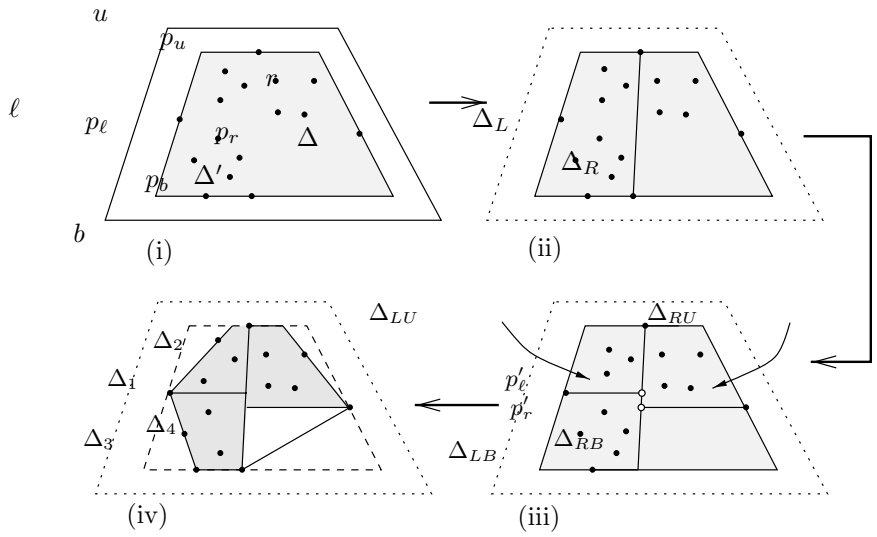


FIG. 4. *Illustration of the canonicalization.*

Δ_1 contain p_u and p_ℓ, respectively, the left edge contains p_ℓ and q_ℓ, and the right edge is determined by the pair of points p_u and p_b. Thus, the trapezoid Δ_1 is in canonical position. The three remaining trapezoids $\Delta_2, \Delta_3, \Delta_4$ are constructed similarly.

In the above construction, if any of the four trapezoids Δ_i does not cover any point of P_Δ (e.g., Δ_4 in Figure 4 (iv)), then we can discard it. Thus, an arbitrary trapezoid of the partition $\mathbf{\Delta}$ can be transformed into at most four canonical trapezoids. This completes the proof of the lemma. □

4. Greedy algorithms. At this point, we can obtain a weak approximation result using the canonical trapezoidal partition. Roughly speaking, we can use the greedy set-cover heuristic [7, 15], ignoring the disjointness constraint, and then refine the heuristic output to produce disjoint trapezoids. Unfortunately, the last step can increase the complexity of the solution quite significantly.

THEOREM 4.1. *Given a set P of n points in the plane and a boolean constraint function $\mathcal{C}$ satisfying* (3.1) *that can be evaluated in polynomial time, we can compute an $O({K_o}^2 \log^2 n)$-size trapezoidal partition of P respecting $\mathcal{C}$ in polynomial time, where K_o is the size of an optimal trapezoidal partition.*

Proof. Consider the set Ξ of all valid, canonical trapezoids in the plane; the set Ξ has $O(n^6)$ trapezoids. We form the geometric set-system

$$X = (P, \quad \{\, P \cap \Delta \mid \Delta \in \Xi \;\&\; \mathcal{C}(\Delta) = 1 \,\}) .$$

X can be computed by testing whether each $\Delta \in \Xi$ is valid. We compute a set cover of X using the greedy algorithm [12, 15] in polynomial time. The greedy algorithm returns a set R consisting of $O(K_o \log^2 n)$ trapezoids, not necessarily disjoint. In order to produce a disjoint cover, we first compute the arrangement $\mathcal{A}(R)$ of the plane induced by R. Then, we decompose each face of $\mathcal{A}(R)$ into trapezoids by drawing a horizontal segment through each vertex until the segment hits an edge of the arrangement. The resulting partition is a trapezoidal partition of P. The number

of trapezoids in the partition is $O(K_o \log n)$, since the arrangement $\mathcal{A}(R)$ has this size. The total running time of the algorithm is polynomial. □

REMARK. (i) The canonical form of trapezoids is used only to construct a finite family of trapezoids to search for an approximate solution. A direct application of the definition in the previous subsection gives a family of $O(n^6)$ canonical trapezoids. By using a slightly different canonical form, we can reduce the size of canonical triangles to $O(n^4)$.

(ii) One can show that the number of trapezoids produced by the above algorithm is $\Omega({K_o}^2)$ in the worst case.

5. A recursively separable partition. We now develop our main approximation algorithm. The algorithm is based on dynamic programming, and it depends on two key ideas: a *recursively separable* partition and a *compliant* partition.

A trapezoidal partition $\mathbf{\Delta}$ is called *recursively separable* if the following hold:

(i) $\mathbf{\Delta}$ consists of a single trapezoid, or

(ii) there exists a line ℓ such that (i) ℓ does not intersect the interior of any trapezoid in $\mathbf{\Delta}$, (ii) $\mathbf{\Delta}^+ = \mathbf{\Delta} \cap \ell^+$ and $\mathbf{\Delta}^- = \mathbf{\Delta} \cap \ell^-$ are both nonempty, where ℓ^+ and ℓ^- are the two half-planes defined by ℓ, and (iii) each of $\mathbf{\Delta}^+$ and $\mathbf{\Delta}^-$ is recursively separable.

The following key lemma gives an upper bound on the penalty incurred by our approximation algorithm if only recursively separable trapezoidal partitions are used.

LEMMA 5.1. *Let P be a finite set of points in the plane and let $\mathbf{\Delta}$ be a trapezoidal partition of P with k trapezoids. There exists a recursively separable partition $\mathbf{\Delta}^\star$ of P with $O(k \log k)$ trapezoids. In addition, each separating line is either a horizontal line passing through a vertex of $\mathbf{\Delta}$ or a line supporting an edge of a trapezoid in $\mathbf{\Delta}$.*

Proof. We present a recursive algorithm for computing $\mathbf{\Delta}^\star$. Our algorithm is similar to the binary space partition algorithm proposed by Paterson and Yao [18]. We assume that the boundaries of the trapezoids in $\mathbf{\Delta}$ are also pairwise disjoint—this assumption is only to simplify our proof.

At each recursive step of the algorithm, the subproblem under consideration lies in a trapezoid T. (This containing trapezoid may degenerate to a triangle, or it may even be unbounded.) The top and bottom edges of T (if they exist) pass through the vertices of $\mathbf{\Delta}$, while the left and right edges (if they exist) are portions of edges of $\mathbf{\Delta}$. Initially T is set to an appropriately large trapezoid containing the family $\mathbf{\Delta}$. Let $\mathbf{\Delta}_T$ denote the trapezoidal partition of $P \cap T$ obtained by intersecting $\mathbf{\Delta}$ with T, and let V_T be the set of vertices of $\mathbf{\Delta}_T$ lying in the interior of T. An edge of $\mathbf{\Delta}_T$ cannot intersect the left or right edge of T, because they are portions of the edge of T. Therefore, each edge of $\mathbf{\Delta}_T$ either lies in the interior of T or intersects only the top and bottom edges of T.

If $|\mathbf{\Delta}_T| = 1$, we return $\mathbf{\Delta}_T$ and stop. Otherwise, we proceed as follows. If there is a trapezoid $\triangle \in \mathbf{\Delta}_T$ that completely crosses T (that is, its vertices lie on the top and bottom edges of T), then we do the following. If $\triangle$ is the leftmost trapezoid of $\mathbf{\Delta}_T$, then we partition T into two trapezoids T_1, T_2 by drawing a line through the right edge of Δ, so that T_1 contains $\triangle$ and T_2 contains the remaining trapezoids of Δ_T. If $\triangle$ is not the leftmost trapezoid of $\mathbf{\Delta}_T$, then we partition T into T_1, T_2 by drawing a line through the left edge of Δ.

If every trapezoid Δ in $\mathbf{\Delta}_T$ has at least one vertex in the interior of T, and so the previous condition is not met, then we choose a point $v \in V_T$ with a median y-coordinate. We partition T into trapezoids T_1, T_2 by drawing a horizontal line ℓ_v passing through v. Each trapezoid $\Delta \in \mathbf{\Delta}_T$ that crosses ℓ_v is partitioned into two

trapezoids by adding the segment $\Delta \cap \ell_v$. At the end of this dividing step, let $\mathbf{\Delta}_1$ and $\mathbf{\Delta}_2$ be the set of trapezoids that lie in T_1 and T_2, respectively. We recursively refine $\mathbf{\Delta}_1$ and $\mathbf{\Delta}_2$ into separable partitions $\mathbf{\Delta}_1^\star$ and $\mathbf{\Delta}_2^\star$, respectively, and return $\mathbf{\Delta}_T^\star = \mathbf{\Delta}_1^\star \cup \mathbf{\Delta}_2^\star$. This completes the description of the algorithm.

We now prove that $\mathbf{\Delta}^*$ satisfies the properties claimed in the lemma. It is clear that $\mathbf{\Delta}^*$ is recursively separable and that each separating line of $\mathbf{\Delta}^*$ either supports an edge of $\mathbf{\Delta}$ or is horizontal. To bound the size of $\mathbf{\Delta}^*$, we charge each trapezoid of $\mathbf{\Delta}^*$ to its bottom-left vertex. Each such vertex is either a bottom-left vertex of a trapezoid of $\mathbf{\Delta}$ or it is an intersection point of a left edge of a trapezoid of $\mathbf{\Delta}$ with the extension of a horizontal edge of another trapezoid of $\mathbf{\Delta}$. There are only k vertices of the first type, so it suffices to bound the number of vertices of the second type. Since the algorithm extends a horizontal edge of a trapezoid of $\mathbf{\Delta}_T$ only if every trapezoid of $\mathbf{\Delta}_T$ has at least one vertex in the interior of T, and we always extend a horizontal edge with a median y-coordinate, it is easily seen that the number of vertices of the second type is $O(k \log k)$. This completes the proof. □

REMARK. Given a family $\mathbf{\Delta}$ of k pairwise disjoint orthogonal rectangles partitioning P, we can find a set of $O(k)$ recursively separable rectangles that forms a *rectangular* partition of P—this uses the *orthogonal* binary space partition algorithm of Paterson and Yao [19].

6. An approximation algorithm. Lemma 5.1 applies to any trapezoidal partition of P. In particular, if we start with a canonical trapezoidal partition $\mathbf{\Delta}$, then the output partition $\mathbf{\Delta}^\star$ is both canonical and recursively separable, and each separating line in $\mathbf{\Delta}^\star$ belongs to the family of canonical lines $\mathcal{L}(P)$. For lack of a better term, we call a trapezoidal partition of P that satisfies these conditions a *compliant partition*. Lemmas 3.1 and 5.1 together imply the following useful theorem.

THEOREM 6.1. *Let P be a set of n points in the plane and let $\mathcal{C}$ be a boolean constraint function satisfying the condition* (3.1). *If there is a trapezoidal partition of P respecting $\mathcal{C}$ with k trapezoids, then there is a compliant partition of P also respecting $\mathcal{C}$ with $O(k \log k)$ trapezoids.*

In the remainder of this section, we give a polynomial-time algorithm, using dynamic programming, for constructing an optimal compliant partition. By Theorem 6.1, this partition has $O(K_o \log K_o)$ trapezoids. Recall that the set $\mathcal{L} = \mathcal{L}(P)$ consists of all canonical lines determined by P.

Consider a subset of points $R \subseteq P$ and a canonical trapezoid Δ containing R. Let $\sigma(R, \Delta)$ denote the size of an optimal compliant partition of R in Δ; the size of a partition is the number of trapezoids in the partition. Theorem 6.1 gives the following recursive definition of σ:

$$\sigma(R, \Delta) = \begin{cases} 1 & \text{if } \mathcal{C}(\Delta) = 1, \\ \min_{\ell} \{\sigma(R^+, \Delta^+) + \sigma(R^-, \Delta^-)\} & \text{otherwise,} \end{cases}$$

where the minimum is over all those lines $\ell \in \mathcal{L}$ that are either horizontal and intersect Δ or intersect both the top and bottom edges of Δ; ℓ^+ and ℓ^- denote the positive and negative half-planes induced by ℓ, $R^+ = R \cap \ell^+$ and $R^- = R \cap \ell^-$. The goal of our dynamic programming algorithm is to compute $\sigma(P, T)$ for some canonical trapezoid T enclosing all the points P. We now describe how the dynamic programming systematically computes the required partial answers.

Every canonical trapezoid Δ in the plane can be described (uniquely) by a 6-tuple $(i, j, k_1, k_2, l_1, l_2)$ consisting of integers between 1 and n. The first two numbers fix

two points p_i and p_j through which the lines containing the top and bottom edges of Δ pass; the second pair fixes the points p_{k_1}, p_{k_2} through which the line containing the left edge of Δ passes; and the third pair fixes the points p_{l_1}, p_{l_2} through which the line containing the right edge of Δ passes. (In case of ties, we may choose the points closest to the corners of Δ.) We use the notation $\Delta(i, j, k_1, k_2, l_1, l_2)$ for the trapezoid associated with the 6-tuple $(i, j, k_1, k_2, l_1, l_2)$. If the 6-tuple does not produce a trapezoid, then $\Delta(i, j, k_1, k_2, l_1, l_2)$ is undefined.

Let $P(i, j, k_1, k_2, l_1, l_2) = P \cap \Delta(i, j, k_1, k_2, l_1, l_2)$. We use the abbreviated notation $\sigma(i, j, k_1, k_2, l_1, l_2)$ to denote the size of an optimal compliant partition for the points contained in $\Delta(i, j, k_1, k_2, l_1, l_2)$:

$$\sigma(i, j, k_1, k_2, l_1, l_2) = \sigma\Big(P(i, j, k_1, k_2, l_1, l_2),\ \Delta(i, j, k_1, k_2, l_1, l_2)\Big).$$

The quantity $\sigma(i, j, k_1, k_2, l_1, l_2)$ is undefined if the trapezoid $\Delta(i, j, k_1, k_2, l_1, l_2)$ is undefined. If the points in P are sorted in increasing order of their y-coordinates, then $\Delta(i, j, k_1, k_2, l_1, l_2)$ is defined only for $i \geq j$. Our dynamic programming algorithm computes the σ values as follows.

If $\mathcal{C}(\Delta(i, j, k_1, k_2, l_1, l_2)) = 1$, then

$$\sigma(i, j, k_1, k_2, l_1, l_2) = 1\,.$$

Otherwise,

$$\sigma(i, j, k_1, k_2, l_1, l_2) \;=\; \min \left\{ \begin{array}{l} \min\limits_{i \leq u \leq j} \{\sigma(i, u, k_1, k_2, l_1, l_2) + \sigma(u+1, j, k_1, k_2, l_1, l_2)\} \\ \min\limits_{v_1, v_2} \{\sigma(i, j, k_1, k_2, v_1, v_2) + \sigma(i, j, v_1, v_2, l_1, l_2)\} \end{array} \right\},$$

where the last minimum varies over all pairs of points v_1, v_2 such that the line passing through them intersects both the top and the bottom edges of $\Delta(i, j, k_1, k_2, l_1, l_2)$.

If Ξ denotes the set of all canonical trapezoids, then $|\Xi| = O(n^6)$; each 6-tuple is associated with at most one unique trapezoid. If $Q(n)$ denotes the time to decide whether $\mathcal{C}(\Delta) = 1$ for an arbitrary trapezoid Δ, then we can initially compute all trapezoids for which $\mathcal{C}(\Delta) = 1$ in total time $O(n^6 Q(n))$; for these trapezoids, we initially set $\sigma(\Delta) = 1$. For all the remaining trapezoids in Ξ, we use the recursive formula presented above to compute their σ. Computing σ for a trapezoid requires computing the minimum of $O(n^2)$ quantities. Thus the total running time of the algorithm is $O(n^8)$. The following theorem states the main result of our paper.

THEOREM 6.2. *Given a set P of n points in the plane and a boolean constraint $\mathcal{C}$ satisfying condition* (3.1)*, we can compute a geometric partition of P with respect to $\mathcal{C}$ using $O(K_o \log K_o)$ trapezoids, where K_o is the number of trapezoids in an optimal partition. Our algorithm runs in worst-case time $O(n^8 + n^6 Q(n))$, where $Q(n)$ is the time to decide whether $\mathcal{C}(R) = 1$ for any subset $R \subseteq P$.*

REMARK. By computing σ's in a more clever order and exploiting certain geometric properties of a geometric partition, the time complexity of the above algorithm can be improved by one order of magnitude. This minor improvement, however, doesn't seem worth the effort needed to explain it.

Theorem 6.2 immediately implies polynomial-time approximation algorithms for the surface-approximation and the planar bichromatic partition problem. In the case of the surface-approximation problem, deciding $\mathcal{C}(\Delta)$ for a trapezoid Δ requires checking whether there is a plane π in $\mathbb{R}^3$ such that the vertical distance between π and the points covered by Δ is at most ε. This problem can be solved in linear time using the

fixed-dimensional linear programming algorithm of Megiddo [16]. (A more practical algorithm, running in time $O(n \log n)$, is the following. Let $A \subseteq P$ denote the set of points covered by the trapezoid Δ. For a point $p \in A$, let p^+ and p^-, respectively, denote the point p translated vertically up and down by ε. Let $A^+ = \{p^+ \mid p \in A\}$ and $A^- = \{p^- \mid p \in A\}$. Then, $\mathcal{C}(\Delta) = 1$ if and only if sets A^+ and A^- can be separated by a plane. The two sets are separable if their convex hulls are disjoint. This can be tested in $O(n \log n)$ time; for instance, see the book by Preparata and Shamos [20].

THEOREM 6.3. *Given a set S of n points in $\mathbb{R}^3$ and a parameter $\varepsilon > 0$, in polynomial time we can compute an ε-approximate polyhedral terrain Σ with $O(K_o \log K_o)$ vertices, where K_o is the number of vertices in an optimal terrain. Our algorithm runs in $O(n^8)$ worst-case time.*

In the planar bichromatic partition problem, deciding whether $\mathcal{C}(\Delta) = 1$ requires checking whether Δ contains any point from the red set R. This can clearly be done in $O(n)$ time. Alternatively, R can be preprocessed in time $O(n^{2+\delta} \log^{O(1)} n)$, for any $\delta > 0$, into a triangle range-searching data structure, which can determine in $O(\log n)$ time whether a query trapezoid contains any point of R; see, e.g., [6]. Our main purpose, however, is to only establish a polynomial-time bound for the approximation algorithm.

THEOREM 6.4. *Given a set R of "red" points and another set B of "blue" points in the plane, we can find in polynomial time a set of $O(K_o \log K_o)$ pairwise disjoint triangles that cover B but do not contain any red point; K_o is the number of triangles in an optimal solution.*

In view of the remark following Theorem 6.1, given a set R of "red" points and another set B of "blue" points in the plane, we can find in polynomial time a set of $O(K_o)$ disjoint orthogonal rectangles that cover B but do not contain any red point. In this case, the time complexity improves by a few orders of magnitude, because there are only $O(n^4)$ canonical rectangles and each rectangle is subdivided into two rectangles by drawing a horizontal or vertical line passing through one of the points. Omitting all the details, the running time in this case is $O(n^5)$. Hence, we obtain the following result.

THEOREM 6.5. *Given a set R of "red" points and another set B of "blue" points in the plane, we can find in polynomial time a set of $O(K_o)$ pairwise disjoint orthogonal rectangles that cover B but do not contain any red point; K_o is the number of rectangles in an optimal solution.*

We conclude this section by noting that Theorem 6.4 can be used to construct small size linear decision trees in the plane. A *linear decision tree* in $\mathbb{R}^d$ is a full binary tree, of which each leaf is labeled $+1$ or -1 and each interior node v is associated with a $(d-1)$-hyperplane h_v; the left (resp., right) child of v is also associated with the half-space lying above (resp., below) h_v. T is used to classify a point $p \in \mathbb{R}^d$ by traversing a path of T, starting from the root, as follows. Suppose v is the node being currently visited. If v is a leaf, return the label of v. Otherwise, if p lies above (resp., below) h_v, we recursively visit the left (resp., right) child of v. T is *consistent* with B and R if it assigns $+1$ to all the points of B and -1 to all the points of R. Each leaf of T corresponds to a convex polyhedron, which is the intersection of the half-spaces associated with its ancestors, and these regions induce a convex partition of $\mathbb{R}^d$. For $d \leq 3$, a convex polyhedron can be triangulated into a linear number of simplices. Hence, for $d \leq 3$, the size of an optimal linear decision tree that is consistent with R and B is $\Omega(K_o)$, where K_o is the size of an optimal bichromatic partition for R and B. See [21] for a survey of known results on decision trees.

Using Theorem 6.4, we can construct a linear decision tree of size $O(K_o \log K_o)$ for R and B as follows. Let $\mathbf{\Delta}$ be the partition computed by the above algorithm. If $\mathbf{\Delta}$ consists of a single trapezoid, say τ, then we construct a linear decision tree with four interior nodes, each associated with the line supporting one of the edges of τ. The leaves of the tree partition the plane into five regions, of which one is τ itself. The leaf corresponding to τ is labeled $+1$ and the others are labeled -1. If $|\mathbf{\Delta}| > 1$, then there is a separating line ℓ; let $\mathbf{\Delta}^+$ (resp., $\mathbf{\Delta}^-$) be the subset of trapezoids lying above (resp., below) ℓ. We associate the root with ℓ, recursively construct the decision trees for $\mathbf{\Delta}^+$ and $\mathbf{\Delta}^-$, and attach them as the left and right subtrees of the root. Putting everything together, we obtain the following result.

THEOREM 6.6. *Given a set R of "red" points and another set B of "blue" points in the plane, we can construct in polynomial time a linear decision tree of size $O(K_o \log K_o)$, which is consistent with R and B; K_o is the size of an optimal linear decision tree that is consistent with R and B.*

7. Extensions. We can extend our algorithm to a slightly stronger form of surface approximation. In the basic problem, the implicit function (surface) is represented by a set of sample points S. What if the sample consists of two-dimensional compact, connected pieces? In this section, we show an extension of our algorithm that deals with the case when the sample consists of a set $\mathcal{T}$ of n *horizontal triangles* with pairwise disjoint xy-projection. (Since any polygon can be decomposed into triangles, this case also handles polygons.) Our goal is to compute a polyhedral terrain Σ, so that the vertical distance between any point in $T_i \in \mathcal{T}$ and Σ is at most ε. We produce a terrain Σ with $O(K_o \log K_o)$ vertices, where K_o is again the number of vertices in an optimal such surface.

Let $\mathcal{T} = \{T_1, \ldots, T_n\}$ be the input set of n horizontal triangles in $\mathbb{R}^3$ with the property that their vertical projections on the plane $z = 0$ are pairwise disjoint. We will consistently use the following notational convention: for an object $s \in \mathbb{R}^3$, $\bar{s}$ denotes its orthogonal projection on the plane $z = 0$, and for a subset $g \subseteq \bar{s}$, $\hat{g}$ denotes the portion of s in $\mathbb{R}^3$ such that $g = \bar{\hat{g}}$. Abusing the notation slightly, we say that a set Ξ of trapezoids (or triangles) in $\mathbb{R}^3$ *ε-approximates* $\mathcal{T}$ within a region $Q \subseteq \mathbb{R}^2$ if the vertical distance between $\mathcal{T}$ and Ξ in Q is at most ε and the vertical projections of trapezoids of Ξ are pairwise disjoint on $z = 0$.

Let S denote the set of vertices of the triangles in $\mathcal{T}$, and let $\bar{S}$ be their orthogonal projection on $z = 0$. We set $P = \bar{S}$, as the set of points in our abstract problem. The constraint function is defined as follows. Given a trapezoid $\Delta \in \mathbb{R}^3$, we have $\mathcal{C}(\Delta) = 1$ if and only if the vertical distance between Δ and any point in $\bigcup_{i=1}^{n} T_i$ directly above/below Δ is at most ε. (Thus, while the point set P includes only the vertices of $\mathcal{T}$, the constraint set takes into consideration the whole triangles.) The constraint $\mathcal{C}$ satisfies (3.1), and it can be computed in polynomial time.

It is also clear that the size of an optimal trapezoidal partition of P with respect to $\mathcal{C}$ is a lower bound on the size of a similar partition for $\mathcal{T}$, the set of triangles. We first apply Theorem 6.3 to obtain a family $\mathbf{\Delta}$ of $O(k \log k)$ trapezoids that ε-approximates P with respect to $\mathcal{C}$; clearly $k \leq K_o$. The next step of our algorithm is to extend $\mathbf{\Delta}$ to a polyhedral terrain that ε-approximates the triangles of $\mathcal{T}$. Care must to be exercised in this step if one wants to add only $O(k \log k)$ new trapezoids. In the second step, we work with the projection of $\mathcal{T}$ and Δ in the plane $z = 0$.

Let $\bar{\mathbf{\Delta}} = \{\bar{\Delta} \mid \Delta \in \mathbf{\Delta}\}$ and $\bar{\mathcal{T}} = \{\bar{T} \mid T \in \mathcal{T}\}$. Let R be the set of connected components of the closure of $\bigcup_{T \in \mathcal{T}} \bar{T} \setminus \bigcup_{\Delta \in \mathbf{\Delta}} \bar{\Delta}$. That is, R is the portion of $\bigcup_{T \in \mathcal{T}} \bar{T}$ lying in the common exterior of $\bar{\mathbf{\Delta}}$, as shown in Figure 5 (i). R is a collection

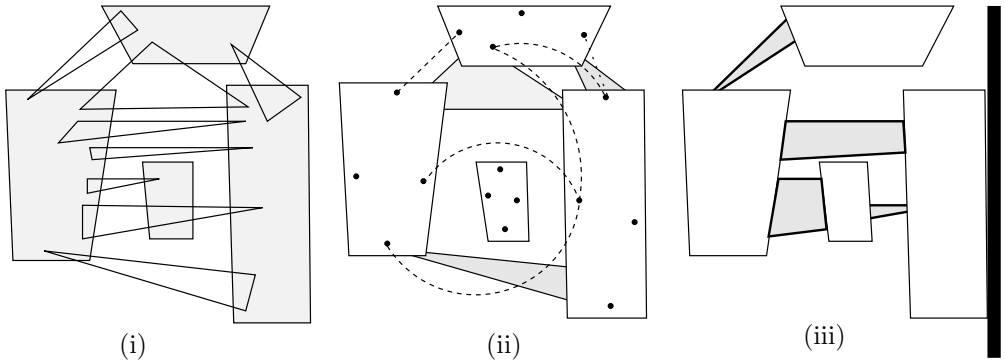


FIG. 5. (i) $\bar{\mathcal{T}}$ *and* $\bar{\mathbf{\Delta}}$*;* (ii) R_1 *and* G*;* (iii) R_2 *and* Q_i*'s.*

of *simple* polygons, each of which is contained in a triangle of $\bar{\mathcal{T}}$. Since the corners of the triangles of $\bar{\mathcal{T}}$ are covered by $\bar{\mathbf{\Delta}}$, the vertices of all polygons in R lie on the boundary of $\bar{\mathbf{\Delta}}$, and each edge of R is contained in an edge of $\bar{\mathbf{\Delta}}$ or of $\bar{\mathcal{T}}$. Let $R_1 \subseteq R$ be the subset of polygons that touch at least three edges of trapezoids in $\bar{\mathbf{\Delta}}$, and let $R_2 = R - R_1$.

For each polygon $P_i \in R_1$, we compute a set of triangles that ε-approximate $\mathcal{T}$ within P_i. For a vertex $v \in P_i$, lying on the boundary of a trapezoid $\bar{\Delta}$, let $\hat{v}$ denote the point on Δ whose xy-projection is v. Let $\bar{T}_i$ be the triangle containing P_i. We triangulate P_i and, for each triangle $\triangle abc$ in the triangulation, we pick $\triangle \hat{a}\hat{b}\hat{c}$. Since T_i is parallel to the xy-plane, it can be proved that the maximum vertical distance between $\triangle \hat{a}\hat{b}\hat{c}$ and T_i is ε. We repeat this step for all polygons in R_1. The number of triangles generated in this step is proportional to the number of vertices in the polygons of R_1, which we bound in the following lemma.

LEMMA 7.1. *The total number of vertices in the polygons of* R_1 *is* $O(k \log k)$.

Proof. Each vertex of a polygon in R_1 is either (i) a vertex of a trapezoid in $\bar{\mathbf{\Delta}}$, or (ii) an intersection point of an edge of $\bar{\mathbf{\Delta}}$ with an edge of a triangle in $\mathcal{T}$. There are only $O(k \log k)$ vertices of type (i), so it remains to bound the number of vertices of type (ii).

We construct an undirected graph $G = (V, E)$, as follows. Let $\Gamma = \{\gamma_1, \ldots, \gamma_t\}$ be the set of edges in $\bar{\mathbf{\Delta}}$.[1] To avoid confusion, we will call the edges of Γ *segments* and those of E *arcs*. For each segment γ_i, we place a point i close to γ_i, inside the trapezoid bounded by γ_i. The set of resulting points forms the node set V. If there is an edge pq of a polygon in R_1 such that $p \in \gamma_i$ and $q \in \gamma_j$, we add the arc (i, j) to E; see Figure 5 (ii). It is easily seen that G is a planar graph and that $|E| = O(k \log k)$. Fix a pair of segments $\gamma_1, \gamma_2 \in \Gamma$ such that $(1, 2) \in E$. Let $E_{12} = \{p_1q_1, p_2q_2, \ldots\}$ be the set of edges in R_1, sorted either left to right or top to bottom, as the case may be, that are incident to γ_1 and γ_2. Let $|E_{12}| = m_{ij}$. Assume that for every $1 \leq i \leq m_{ij}$, p_i lies on γ_1 and q_i lies on γ_2. The number of vertices of type (ii) is obviously $2\sum_{(i,j)\in E} m_{ij}$. We call two edges $p_iq_i,\ p_jq_j \in E_{12}$ *equivalent* if the interior of the convex hull of p_iq_i and p_jq_j does not intersect any trapezoid of $\bar{\mathbf{\Delta}}$. This equivalence relation partitions E_{12} into equivalent classes, each consisting of a contiguous subsequence of E_{12}. Let μ_{ij} denote the number of equivalence classes in E_{ij}.

[1] The segments of Γ may overlap, because the trapezoids of $\bar{\mathbf{\Delta}}$ can touch each other. If a segment γ_i of Γ is an edge of two trapezoids, then no edge of R_1 can be incident to γ_i.

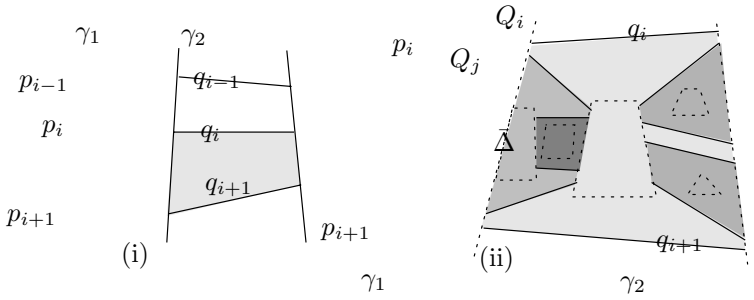


FIG. 6. (i) *Edges in one equivalent class of* E_{12}*, and* (ii) *edges in different equivalent classes.*

CLAIM 1. *There are at most two edges in each equivalence class of* E_{12}*.*

Proof. Assume for the sake of a contradiction that three edges $p_{i-1}q_{i-1}$, p_iq_i, and $p_{i+1}q_{i+1}$ belong to the same equivalence class. Further, assume that the triangle $\bar{T} \in \bar{\mathcal{T}}$ bounded by p_iq_i lies below p_iq_i (see Figure 6 (i)). Since the quadrilateral Q defined by p_iq_i and $p_{i+1}q_{i+1}$ does not contain any trapezoid of $\bar{\mathbf{\Delta}}$, $p_{i+1}q_{i+1}$ is also an edge of $\bar{T}$ and $Q \subseteq \bar{T}$. But then Q is a connected component of R and it touches only two edges of $\bar{\mathbf{\Delta}}$, thereby implying that p_iq_i is not an edge of a polygon of R_1, a contradiction. ∎

Thus,

$$\sum_{(i,j)\in E} m_{ij} \;\le\; 2\sum_{(i,j)\in E} \mu_{ij} \;=\; 2|E| + \sum_{(i,j)\in E} (\mu_{ij}-1) \;=\; O(k\log k) + \sum_{(i,j)\in E} (\mu_{ij}-1)\,.$$

Next, we bound the quantity $\sum_{(i,j)\in E}(\mu_{ij}-1)$. Let E_{12}^j and E_{12}^{j+1} be two consecutive equivalence classes of E_{12}, let p_iq_i be the bottom edge of E_{12}^j, and let $p_{i+1}q_{i+1}$ be the top edge of E_{12}^{j+1}. The quadrilateral $Q_{12}^j = p_iq_iq_{i+1}p_{i+1}$ contains at least one trapezoid $\bar{\Delta}$ of $\bar{\mathbf{\Delta}}$. We call $p_iq_i, p_{i+1}q_{i+1}$ the *triangle edges* of Q, and p_ip_{i+1}, q_iq_{i+1} the *trapezoidal edges* of Q. The triangle edges of Q_{12}^j are adjacent in E_{12}. Let

$$\mathcal{Q} = \bigcup \{Q_{ij}^l \mid (i,j) \in E \text{ and } l < \mu_{ij}\}$$

be the set of resulting quadrilaterals. Since $|\mathcal{Q}| = \sum_{(i,j)\in E}(\mu_{ij}-1)$, it suffices to bound the number of quadrilaterals in $\mathcal{Q}$.

Consider the planar subdivision induced by $\mathcal{Q}$ and call it $\mathcal{A}(\mathcal{Q})$. The number of faces in the subdivision is proportional to the number of quadrilaterals in $\mathcal{Q}$, so we bound the number of faces in the subdivision. For each bounded face $f \in \mathcal{A}(\mathcal{Q})$, let $Q(f)$ be the smallest quadrilateral of $\mathcal{Q}$ that contains f. Since the boundaries of quadrilaterals do not cross, $Q(f)$ is well defined.

CLAIM 2. *Every face* f *of* $\mathcal{A}(\mathcal{Q})$ *can be uniquely associated with a trapezoid* $\Delta \in \bar{\mathbf{\Delta}}$ *such that* $\Delta \subseteq f$*.*

Proof. The claim is obviously true for the unbounded face, so assume that f is a bounded face. If $Q_i = Q(f)$ does not contain any other quadrilateral of $\mathcal{Q}$, then $f = Q_i$. As mentioned above, Q_i, and therefore f, contains a trapezoid of $\bar{\mathbf{\Delta}}$. We associate this trapezoid with f. If there is more than one trapezoid lying in f, we arbitrarily choose one of them.

If $Q_i = Q(f)$ contains another quadrilateral of $\mathcal{Q}$, let $Q_j \in \mathcal{Q}$ be the largest quadrilateral that lies inside Q_i; that is, ∂Q_j is a part of ∂f. If neither of the

trapezoidal edges of Q_j lies in the interior of Q_i, then the trapezoidal edges of both Q_i, Q_j lie on the same segments of Γ, say, γ_1, γ_2. Consequently, the triangle edges of both Q_i, Q_j belong to E_{12}, which is impossible, because then the triangle edges of Q_i are not adjacent in E_{12}. Hence, one of the trapezoidal edges of Q_j lies in the interior of Q_i. Let $\bar{\Delta}$ be the trapezoid bounded by this edge. Since the triangle edges of Q_j lie outside $\bar{\Delta}$ and the interior of $\bar{\Delta}$ does not intersect any edge of R_1, $\bar{\Delta}$ lies in f. We associate $\bar{\Delta}$ with f. It can be verified that $\bar{\Delta}$ is not associated with any other face. This completes the proof of Claim 2. □

By Claim 2, the number of faces in $\mathcal{A}(\mathcal{Q})$ is at most $|\bar{\mathbf{\Delta}}| = O(k \log k)$. Hence,

$$\sum_{(i,j)\in E} (\mu_{ij} - 1) = |\mathcal{Q}| = O(k \log k),$$

as required. □

Next, we partition the polygons of R_2 into equivalence classes in the same way as we partitioned the edges of E_{12} in the proof of Lemma 7.1. That is, we call two polygons $\tau_1, \tau_2 \in R_2$ *equivalent* if (i) their endpoints lie on the same pair of edges in $\bar{\mathbf{\Delta}}$, and (ii) the interior of the convex hull of $\tau_1 \cup \tau_2$ does not intersect any trapezoid of $\bar{\mathbf{\Delta}}$. Using the same argument as in the proof of the above lemma, the following lemma can be established.

LEMMA 7.2. *The edges of R_2 can be partitioned into $O(k \log k)$ equivalence classes.*

For each equivalence class $E_i \subseteq R_2$, let Q_i be the convex hull of E_i—observe that Q_i is a convex quadrilateral, as illustrated in Figure 5 (iii). Each quadrilateral Q_i can be ε-approximated using at most three triangles in $\mathbb{R}^3$ in the same way as we approximated each polygon P_i of R_1. By Lemma 7.2, the total number of triangles created in this step is also $O(k \log k)$.

Putting together these pieces, we obtain the following lemma.

LEMMA 7.3. *The family of trapezoids $\mathbf{\Delta}$ can be supplemented with $O(k \log k)$ additional trapezoids in $\mathbb{R}^3$ so that all the triangles of $\mathcal{T}$ are ε-approximated. The orthogonal projection of all the trapezoids on the plane $z = 0$ is pairwise disjoint.*

The area not covered by the projection of trapezoids found in the preceding lemma, of course, can be approximated without any regard to the triangles of $\mathcal{T}$. The final surface has $O(K_o \log K_o)$ trapezoids and it ε-approximates the family of triangles $\mathcal{T}$. We finish with a statement of our main theorem in this section.

THEOREM 7.4. *Given a set of n horizontal triangles in $\mathbb{R}^3$, with pairwise disjoint projection on the plane $z = 0$ and a parameter $\varepsilon > 0$, we can compute in polynomial time an ε-approximate polyhedral terrain of size $O(K_o \log K_o)$ for $\mathcal{T}$, where K_o is the size of an optimal ε-approximate terrain.*

8. Closing remarks. We presented an approximation technique for certain geometric covering problems with a disjointness constraint. Our algorithm achieves a logarithmic performance guarantee on the size of the cover, thus matching the bound achieved by the "greedy set cover" heuristic for arbitrary sets and no disjointness constraint. Applications of our result include polynomial-time algorithms to approximate a monotone, polyhedral surface in 3-space, and to approximate the disjoint cover by triangles of red-blue points. We also proved that these problems are *NP-hard*.

The surface-approximation problem is an important problem in visualization and computer graphics. The state of theoretical knowledge on this problem appears to be rather slim. Except for the convex surfaces, no approximation algorithms with good performance guarantees are known [7, 17]. For the approximation of convex

polytopes, it turns out that one does not need *disjoint* covering, and therefore the greedy set-cover heuristic works.

We conclude by mentioning some open problems. An obvious open problem is to reduce the running time of our algorithm for it to be of any practical value. Finding efficient heuristics with good performance guarantees seems hard for most of the geometric partitioning problems and requires further work. A second problem of great practical interest is to ε-approximate general polyhedra—this problem arises in many real applications of computer modeling. To the best of our knowledge, the latter problem remains open even for the special case where one wants to find a minimum-vertex polyhedral surface that lies between two monotone surfaces. The extension of our algorithm presented in Section 7 does not work because we do not know how to handle the last fill-in stage.

Acknowledgments. The authors thank Joe Mitchell, Prabhakar Raghavan, and John Reif for several helpful discussions.

REFERENCES

[1] E. Allgower and S. Gnutzmann, *An algorithm for piecewise linear approximation of an implicitly defined two-dimensional surfaces,* SIAM J. Numer. Anal., 24 (1987), pp. 452–469.

[2] E. Allgower and P. Schmidt, *An algorithm for piecewise linear approximation of an implicitly defined manifold,* SIAM J. Numer. Anal., 22 (1985), pp. 322–346.

[3] E. Arkin, H. Meijer, J. Mitchell, D. Rappaport, and S. Skiena, *Decision trees for geometric models,* in Proc. 9th Annual ACM Symp. on Comput. Geom., 1993, pp. 369–378.

[4] J. Bloomenthal, *Polygonization of implicit surfaces,* Computer Aided Design, 5 (1988), pp. 341–355.

[5] H. Brönnimann and M. T. Goodrich, *Almost optimal set covers in finite VC-dimension,* Discrete Comput. Geom., 14 (1995), pp. 263–279.

[6] B. Chazelle, M. Sharir, and E. Welzl, *Quasi-optimal upper bounds for simplex range searching and new zone theorems,* Algorithmica, 8 (1992), pp. 407–429.

[7] K. L. Clarkson, *Algorithms for polytope covering and approximation,* in Proc. 3rd Workshop on Algorithms and Data Structures, Lectures Notes in Comput. Sci. 709, Springer-Verlag, New York, 1993, pp. 246–252.

[8] M. DeHaemer and M. Zyda, *Simplification of objects rendered by polygonal approximations,* Computers and Graphics, 15 (1992), pp. 175–184.

[9] S. Fekete, *Several Hardness Results on Problems of Point Separation and Line Stabbing,* Tech. Report, Dept. of Applied Math. and Stats., SUNY Stony Brook, New York, 1993.

[10] R. Fowler, M. Paterson, and L. Tanimoto, *Optimal packing and covering in the plane are NP-complete,* Inform. Process. Lett., 35 (1981), pp. 85–92.

[11] I. Ihm and B. Naylor, *Piecewise linear approximations of digitized space curves with applications*, in Scientific Visual. of Physical Phenomena, Springer-Verlag, New York, 1991, pp. 545–569.

[12] D. S. Johnson, *Approximation algorithms for combinatorial problems,* J. Computer Systems Sci., 9 (1974), pp. 256–278.

[13] D. Lichtenstein, *Planar formulae and their uses,* SIAM J. Comput., 11 (1982), pp. 329–343.

[14] W. Lorenson and H. Cline, *Marching cubes: A high resolution 3D surface construction algorithm,* Computer Graphics, 21 (1987), pp. 163–169.

[15] L. Lovász, *On the ratio of optimal integral and fractional cover,* Discrete Math., 13 (1975), pp. 383–390.

[16] N. Megiddo, *Linear-time algorithms for linear programming in* $\mathbb{R}^3$ *and related problems,* SIAM J. Comput., 12 (1983), pp. 759–776.

[17] J. Mitchell and S. Suri, *Separation and approximation of polyhedral surfaces,* Comput. Geom. Theory Appl., 5 (1995), pp. 95–114.

[18] M. S. Paterson and F. F. Yao, *Efficient binary space partitions for hidden-surface removal and solid modeling,* Discrete and Comput. Geom., 5 (1990), pp. 485–503.

[19] M. S. Paterson and F. F. Yao, *Optimal binary space partitions for orthogonal objects,* J. Algorithms, 13 (1992), pp. 99–113.
[20] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.
[21] S. Salzberg, R. Chandar, H. Ford, S. Murthy, and R. White, *A survey of decision tree classifier methodology,* IEEE Trans. Systems, Man and Cybernetics, 21 (1991), pp. 660–674.
[22] W. Schroeder, J. Zarge, and W. Lorensen, *Decimation of triangle meshes,* Computer Graphics, 26 (1992), pp. 65–78.
[23] F. Schmitt, B. Barsky, and W. H. Du, *An adaptive subdivision method for surface fitting from sampled data,* Computer Graphics, 20 (1986), pp. 179–188.
[24] S. Suri, *On some link distance problems in a simple polygon,* IEEE Trans. Robotics and Automation, 6 (1990), pp. 108–113.
[25] G. Turk, *Re-tiling of polygonal surfaces,* Computer Graphics, 26 (1992), pp. 55–64.

SIAM J. COMPUT.
Vol. 27, No. 4, pp. 1036–1072, August 1998
© 1998 Society for Industrial and Applied Mathematics
007

RANDOMIZED DATA STRUCTURES FOR THE DYNAMIC CLOSEST-PAIR PROBLEM*

MORDECAI GOLIN†, RAJEEV RAMAN‡, CHRISTIAN SCHWARZ§, AND MICHIEL SMID¶

Abstract. We describe a new randomized data structure, the *sparse partition*, for solving the dynamic closest-pair problem. Using this data structure the closest pair of a set of n points in D-dimensional space, for any fixed D, can be found in constant time. If a frame containing all the points is known in advance, and if the floor function is available at unit cost, then the data structure supports insertions into and deletions from the set in expected $O(\log n)$ time and requires expected $O(n)$ space. This method is more efficient than any deterministic algorithm for solving the problem in dimension $D > 1$. The data structure can be modified to run in $O(\log^2 n)$ expected time per update in the algebraic computation tree model. Even this version is more efficient than the best currently known deterministic algorithm for $D > 2$. Both results assume that the sequence of updates is not determined in any way by the random choices made by the algorithm.

Key words. computational geometry, proximity, dynamic data structures, randomization

AMS subject classification. 68U05

PII. S0097539794277718

1. Introduction.

We consider the *dynamic closest-pair problem*: we are given an initially empty set S of points in D-dimensional space and want to keep track of the closest pair of points in S, as S is being modified by insertions and deletions of individual points. We assume that D is an arbitrary constant and that distances are measured in the L_t-metric for some fixed t, $1 \le t \le \infty$. Recall that in the L_t-metric, the distance $d_t(p, q)$ between two points $p = (p^{(1)}, \ldots, p^{(D)})$ and $q = (q^{(1)}, \ldots, q^{(D)})$ in D-dimensional space is defined by

$$d_t(p, q) := \left(\sum_{i=1}^{D} |p^{(i)} - q^{(i)}|^t \right)^{1/t}$$

if $1 \le t < \infty$, and for $t = \infty$ it is defined by

$$d_\infty(p, q) := \max_{1 \le i \le D} |p^{(i)} - q^{(i)}|.$$

Throughout this paper, t will be implicit, and we will write $d(p, q)$ for $d_t(p, q)$.

The precursor to this problem is the classical *closest-pair problem* which is to compute the closest pair of points in a static set S, $|S| = n$. Shamos and Hoey [20] and

*Received by the editors November 23, 1994; accepted for publication (in revised form) May 24, 1996; published electronically May 19, 1998. This research was supported by the European Community, Esprit Basic Research Action 7141 (ALCOM II). A preliminary version appeared in the *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms* (*SODA*), 1993. The work was partly done while the first author was employed at INRIA Rocquencourt, France, and visiting Max-Planck-Institut für Informatik, Germany, and the second and the third authors were employed at Max-Planck-Institut für Informatik.

http://www.siam.org/journals/sicomp/27-4/27771.html

†Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong (golin@cs.ust.hk). This author was partially supported by NSF grant CCR-8918152 and by Hong Kong RGC/CRG grant 181/93E.

‡Algorithm Design Group, Department of Computer Science, King's College London, Strand, London WC2R 2LS, UK (raman@dcs.kcl.ac.uk).

§International Computer Science Institute, Berkeley, CA 94704 (schwarz@icsi.berkeley.edu).

¶Max-Planck-Institut für Informatik, D-66123 Saarbrücken, Germany (michiel@mpi-sb.mpg.de).

Bentley and Shamos [2] gave $O(n \log n)$-time algorithms for solving the closest-pair problem in the plane and in arbitrary but fixed dimension, respectively. This running time is optimal in the algebraic computation tree model [1]. If we allow randomization as well as the use of the (nonalgebraic) floor function, we find algorithms with better (expected) running times for the closest-pair problem. Rabin, in his seminal paper [16] on randomized algorithms, gave an algorithm for this problem which ran in $O(n)$ expected time [16, 5]. Since then, alternative methods with the same running time have been discovered. In addition to the randomized incremental algorithm presented in [11], there is a different approach, described by Khuller and Matias [12], which uses a randomized "filtering" procedure. This method is at the heart of our dynamic algorithm.

There has been a lot of work on maintaining the closest pair of a dynamically changing set of points. When we were restricted to the case where only insertions of points are allowed (sometimes known as the *on-line closest-pair problem*), a series of papers culminated in an optimal data structure due to Schwarz, Smid, and Snoeyink [23]. Their data structure required $O(n)$ space and supported insertions in $O(\log n)$ time.

The existing results are not as satisfactory when deletions must be performed. If only deletions are to be performed, Supowit [24] gave a data structure with $O(\log^D n)$ amortized update time which uses $O(n \log^{D-1} n)$ space. When both insertions and deletions are allowed, Smid [22] described a data structure which uses $O(n \log^D n)$ space and runs in $O(\log^D n \log \log n)$ amortized time per update. Another data structure due to Smid [21], with improvements stemming from results of Salowe [17] and Dickerson, Drysdale, and Sack [7], uses $O(n)$ space and requires $O(\sqrt{n} \log n)$ time for updates. Very recently, after a preliminary version of this paper was presented, Kapoor and Smid [13] devised a deterministic data structure of linear size which achieves polylogarithmic amortized update time, namely, $O(\log^{D-1} n \log \log n)$ for $D \geq 3$ and $O(\log^2 n/(\log \log n)^\ell)$ for the case $D = 2$, where ℓ is an arbitrary nonnegative integer constant.

In this paper we discuss a randomized data structure, the *sparse partition,* which solves the dynamic closest-pair problem in arbitrary fixed dimension using $O(\log n)$ expected time per update. The data structure needs $O(n)$ expected space. This time bound is obtained assuming that the floor function can be computed in constant time and that the algorithm has prior knowledge of a frame which contains all the points in S at any time (this assumption enables the algorithm to create and use dynamic hash tables). Without either of the above assumptions, we obtain an $O(\log^2 n)$ expected update time in the algebraic computation tree model. Even this version of the randomized algorithm is more efficient than the currently best-known deterministic algorithms for solving the problem for $D > 2$ and almost matches the running time of the recently developed method of Kapoor and Smid [13] in the case $D = 2$. Indeed, our algorithm is the first to obtain polylogarithmic update time using linear space for the dynamic closest-pair problem.

Our results require that the updates are generated by an *oblivious* adversary, who fixes a worst-case sequence of operations in advance and reveals it to the data structure in an on-line manner. Hence, the adversary's knowledge of the internal state of the data structure (which is a random variable) is limited to that which can be obtained a priori, and, in particular, the sequence of updates is not determined in any way by the random choices made by the algorithm.

Given a set S of points, the sparse partition that stores S will be a randomly

chosen one from many possible structures. In one version of the data structure, the probability that a particular structure is the one that is being used will depend only on the set S that is being stored and not upon the sequence of insertions and deletions that were used to construct S. In this sense, the data structure is reminiscent of skip lists or randomized search trees [15, 19].

The paper is organized as follows. In section 2, we give an implementation-independent definition of the sparse partition, give a static algorithm to build it, and show how to augment this data structure to find the closest pair in constant time. A grid-based implementation of the sparse partition is given in section 3, and algorithms for updating the sparse partition are given in section 4. In section 5, we show how to modify the grid-based data structure in order to obtain an algorithm which fits in the algebraic computation tree model. Section 6 contains extensions of the method. We show how to modify the data structure to achieve $O(n)$ worst-case space rather than that which is only expected, at the cost of making the update time bound amortized. We also give high probability bounds for the running time of a sequence of update operations. Section 7 contains some concluding remarks.

2. Sparse partitions. We start with some notation. Let S be a set of n points in D-dimensional space. The *minimal distance* of S is $\delta(S) := \min\{d(p,q) : p,q \in S,\ p \neq q\}$. A *closest pair* in S is a pair $p,q \in S$ such that $d(p,q) = \delta(S)$. The distance of p to its nearest neighbor in S is denoted by $d(p,S) := \min\{d(p,q) : q \in S \setminus \{p\}\}$. For convenience, we often speak of *the* closest pair, although there might be more than one. This causes no problems since, as we shall see, our data structure not only allows us to find a single closest pair but also to report all pairs of points attaining the minimal distance in time proportional to their number.

We now describe an idealized version of our data structure. Although the final data structure will be different in many significant ways, the idealized description may offer some insight into the working of the final data structure. For now, assume that the points are in general position and define the *sparseness* of a point $p \in S$ as being simply $d(p,S)$. We will assume the existence of a data structure which, for some fixed $\delta > 0$, maintains a set T of points under insertions, deletions, and queries of the form "is $d(p,T) > \delta$?" for an arbitrary point p, and "enumerate all $q \in T$ with $d(p,q) \leq \delta$." Insertions, deletions, and the former query are assumed to take constant time, while the latter query takes constant time per element reported.

Our idealized data structure partitions the points of S based on their sparseness. Suppose the set S initially contains n points, and we preprocess S to obtain a distance threshold δ such that between $1/3$ and $2/3$ of the points have sparseness greater than δ. This preprocessing can be done in $O(n)$ expected time as follows: simply pick a random $p \in S$, compute $\delta = d(p,S)$, and check how many points have sparseness greater than δ. The fundamental observation of [12] is that p's position in a list of the points in S ordered by sparseness is also random, and hence $O(1)$ such random trials suffice on average to compute a suitable threshold.

We now perform a sequence of cn updates to S for some sufficiently small constant $c > 0$. Let S' denote the set of points in S at any time during this sequence of updates whose sparseness is greater than δ. Since the sparseness of a point changes only when its nearest neighbor in S changes, and a point can only be the nearest neighbor of $O(1)$ points in the set, each update can only change the sparseness of $O(1)$ other points. Hence, during this sequence of updates, S' will continue to contain a constant fraction of points in S. Furthermore, no point in S' will be part of a closest pair in S. At the end of this sequence of updates we can recompute a suitable threshold in linear time,

the cost of which can be amortized over the $\Theta(n)$ updates since the last threshold computation. Applying the partitioning idea recursively to the set $S \setminus S'$ then leads, modulo some details, to a solution to the closest-pair problem with $O(\log n)$ expected amortized time per update—in essence, there will be $O(\log n)$ levels of recursion, and each update results in $O(1)$ amortized expected work per level of recursion.

The most significant problem with this idealized approach is that the "distance threshold" queries that we postulated appear difficult to answer in reality. However, we can quite easily answer the *approximate* versions of these queries, which can distinguish between the cases $d(p,T) > (1+\epsilon)\delta$ and $d(p,T) \leq \delta$ for some fixed $\epsilon > 0$ but may give an arbitrary answer for values in between (a grid-based implementation using the floor function will doubtless occur to a reader familiar with Rabin's algorithm [16]). This causes many problems with the above approach which we have not been able to overcome in a direct manner. The main differences are that the sets "sieved out" at each level have to be more carefully defined and that the rebuilding rules now contain probabilistic components rather than being purely deterministic as above. A few more complications arise when we go from the grid-based algorithm to the algebraic one.

We now present an abstract framework which captures all the properties that we need from the sets sieved out at each level and prove some basic facts about these sets in the abstract framework.

DEFINITION 2.1. *Let S be a set of points in D-space. A* sparse partition *for the set S is a sequence of 5-tuples $(S_i, S'_i, p_i, q_i, \delta_i)$, $1 \leq i \leq L$, where L is a positive integer, such that the following hold.*

(a) *For $i = 1, \ldots, L$,*
  (a.1) *$S_i \neq \emptyset$;*
  (a.2) *$S'_i \subseteq S_i \subseteq S$;*
  (a.3) *$p_i, q_i \in S_i$ and $p_i \neq q_i$ if $|S_i| > 1$;*
  (a.4) *$\delta_i = d(p_i, q_i) = d(p_i, S_i)$.*

(b) *For all $1 \leq i \leq L$, and for all $p \in S_i$,*
  (b.1) *if $d(p, S_i) > \delta_i/2$ then $p \in S'_i$;*
  (b.2) *if $d(p, S_i) \leq \delta_i/4D$ then $p \notin S'_i$.*

(c) *For all $1 \leq i < L$ and for all $p \in S_i$,*
  if $p \in S_{i+1}$, then there is a point $q \in S_i$ such that $d(p,q) \leq \delta_i/2$ and $q \in S_{i+1}$.

(d) *$S_1 = S$ and for $1 \leq i \leq L-1$, $S_{i+1} = S_i \setminus S'_i$.*

For each i, we call the points of S'_i the sparse points *in S_i, and the set S'_i the* sparse set. *Each 5-tuple itself is also called a* level *of the partition.*

Conditions (b.1) and (b.2) govern the decision on whether a point of S_i is in the sparse set S'_i or not. The threshold values given in (b.1) and (b.2) depend on the nearest neighbor distance $d(p_i, S_i)$ of the point $p_i \in S_i$, which will be called the *pivot* in the following. For a point $p \in S_i$ such that $d(p_i, S_i)/4D < d(p, S_i) \leq d(p_i, S_i)/2$, the decision may be arbitrary as far as the results of this section are concerned and will be made precise by the implementation later. Condition (c) is used while implementing updates to the sparse partition efficiently. Some readers may now wish to adjourn to section 3 and read up to the end of the proof of Lemma 3.2, where a concrete implementation of a sparse partition is discussed, before continuing with the rest of this section.

LEMMA 2.1. *Any sparse partition for S satisfies the following properties:*

(1) *The sets S'_i, for $1 \leq i \leq L$, are nonempty and pairwise disjoint. For any $1 \leq i \leq L$, $S_i = \bigcup_{j \geq i} S'_j$. In particular, $\{S'_1, S'_2, \ldots, S'_L\}$ is a partition of S.*

(2) *For any* $1 \le i < L$, $\delta_{i+1} \le \delta_i/2$. *Moreover,* $\delta_L/4D < \delta(S) \le \delta_L$.

Proof. Part (1) is obvious. To prove the first part of (2), let $1 \le i < L$. Since $p_{i+1} \in S_{i+1}$, we know from condition (c) in Definition 2.1 that there is a point $q \in S_i$ such that $d(p_{i+1}, q) \le \delta_i/2$ and $q \in S_{i+1}$. Therefore,

$$\delta_{i+1} = d(p_{i+1}, S_{i+1}) \le d(p_{i+1}, q) \le \delta_i/2.$$

To prove the second part of (2), let p, q be a closest pair in S. Let i and j be such that $p \in S'_i$ and $q \in S'_j$. Assume w.l.o.g. that $i \le j$. Then it follows from (1) that p and q are both contained in S_i. It is clear that $\delta(S) = d(p, q) = d(p, S_i)$. Condition (b.2) in Definition 2.1 implies that $d(p, S_i) > \delta_i/4D$, and from the first part of (2), we conclude that $\delta(S) > \delta_i/4D \ge \delta_L/4D$. The inequality $\delta(S) \le \delta_L$ obviously holds, because δ_L is a distance between two points of S. □

DEFINITION 2.2. *Let S_i be some set of points and let $(S_i, S'_i, p_i, q_i, \delta_i)$ be a* 5*-tuple chosen randomly from some distribution. This* 5*-tuple is called* uniform *if, for all $p \in S_i$, $\Pr[p = p_i] = 1/|S_i|$. Now let $(S_i, S'_i, p_i, q_i, \delta_i)$, $1 \le i \le L$, be a sparse partition for the set S chosen randomly from some distribution on all of the sparse partitions on $S_1 = S$. The set S is said to be* uniformly stored *by the sparse partition if all its* 5*-tuples are uniform.*

We now give an algorithm that, given an input set S, stores it uniformly as a sparse partition.

ALGORITHM *Sparse_Partition*(S)

(i) $S_1 := S$; $i := 1$.

(ii) Choose a random point $p_i \in S_i$. Calculate $\delta_i = d(p_i, S_i)$. Let $q_i \in S_i$ be such that $d(p_i, q_i) = \delta_i$.

(iii) Choose S'_i to satisfy (b.1), (b.2), and (c) in Definition 2.1.

(iv) If $S_i = S'_i$ stop; otherwise set $S_{i+1} := S_i \setminus S'_i$, set $i := i + 1$ and goto (ii).

LEMMA 2.2. *Let S be a set of n points in $\mathbb{R}^D$. Run Sparse_Partition(S) and let $(S_i, S'_i, p_i, q_i, \delta_i)$, $1 \le i \le L$, be the* 5*-tuples constructed by the algorithm. Then this set of* 5*-tuples is a uniform sparse partition for S, and we have* $\mathrm{E}(\sum_{i=1}^{L} |S_i|) \le 2n$.

Proof. The output generated by algorithm *Sparse_Partition*(S) obviously fulfills the requirements of Definitions 2.1 and 2.2. To prove the bound on the size of the structure, we first note that $L \le n$ by Lemma 2.1. Define $S_{L+1} := S_{L+2} := \cdots := S_n := \emptyset$. Let $s_i := \mathrm{E}(|S_i|)$ for $1 \le i \le n$. We will show that $s_{i+1} \le s_i/2$, from which it follows that $s_i \le n/2^{i-1}$. By the linearity of expectation, we get $\mathrm{E}(\sum_{i=1}^{L} |S_i|) \le \sum_{i=1}^{n} n/2^{i-1} \le 2n$.

It remains to prove that $s_{i+1} \le s_i/2$. If $s_i = 0$, then $s_{i+1} = 0$ and the claim holds. So assume $s_i > 0$. We consider the conditional expectation $\mathrm{E}(|S_{i+1}| \mid |S_i| = l)$. Let $r \in S_i$ such that $d(r, S_i) \ge \delta_i$. Then, condition (b.1) of Definition 2.1 implies that $r \in S'_i$; i.e., $r \notin S_{i+1}$.

Take the points in S_i and label them $r_1, r_2, \ldots, r_l$ such that $d(r_1, S_i) \le d(r_2, S_i) \le \cdots \le d(r_l, S_i)$. The point p_i is chosen randomly from the set S_i, so it can be any of the r_j's with equal probability. Thus $\mathrm{E}(|S_{i+1}| \mid |S_i| = l) \le l/2$, from which it follows that $s_{i+1} = \sum_l \mathrm{E}(|S_{i+1}| \mid |S_i| = l) \cdot \Pr(|S_i| = l) \le s_i/2$. □

We remark that the procedure *Sparse_Partition* is the essential component of Khuller and Matias's algorithm [12], where it was used as a filtering procedure to compute δ_L and only kept track of the current set S_i. Our dynamic algorithm stores the sets S_i and S'_i at all the levels. As we now show, the minimal distance is closely related to the sparse sets S'_i, i.e., the points that were thrown away in each step of the iteration in [12]. This enables us to use the sparse partition to find $\delta(S)$ quickly.

DEFINITION 2.3. *Let $S'_1, S'_2, \ldots, S'_L$ be the sparse sets of a sparse partition for S. For any $p \in \mathbb{R}^D$ and $1 \le i \le L$, define the* restricted distance

$$d_i^*(p) := \min\left(\delta_i,\, d\big(p,\, \bigcup_{1\le j\le i} S'_j\big)\right),$$

i.e., the smaller of δ_i and the minimal distance between p and all points in $S'_1 \cup S'_2 \cup \cdots \cup S'_i$.

For convenience, we define, for all $i \le 0$, $S'_i := \emptyset$, $\delta_i := \infty$, and $d_i^*(p) := \infty$ for any point p.

LEMMA 2.3. *Let $p \in S$ and let i be the index such that $p \in S'_i$.*

(1) *$d_i^*(p) > \delta_i/4D$.*

(2) *If $q \in S'_j$, where $1 \le j < i - D$, then $d(p,q) > \delta_i$.*

(3) *$d_i^*(p) = \min\big(\delta_i,\, d(p, S'_{i-D} \cup S'_{i-D+1} \cup \cdots \cup S'_i)\big)$.*

Proof. The proof of (1) is obvious. To prove (2), let $q \in S'_j$, where $1 \le j < i - D$. Since $d(p,q) > \delta_j/4D$, Lemma 2.1 implies that

$$d(p,q) > \frac{\delta_j}{4D} \ge \frac{\delta_{i-D-1}}{4D} \ge \frac{2^{D+1}\delta_i}{4D} \ge \delta_i.$$

(3) follows immediately from (2). □

LEMMA 2.4.

$$\delta(S) = \min_{1\le i\le L} \min_{p\in S'_i} d_i^*(p) = \min_{L-D\le i\le L} \min_{p\in S'_i} d_i^*(p).$$

Proof. The value $d_i^*(p)$ is always the distance between two points in S. Therefore, $\delta(S) \le \min_{1\le i\le L} \min_{p\in S'_i} d_i^*(p)$. Let p, q be a closest pair, with $p \in S'_i$ and $q \in S'_j$. Assume w.l.o.g. that $j \le i$. Clearly, $d(p,q) = d(p, \bigcup_{\ell\le i} S'_\ell) \ge d_i^*(p)$. This implies that $\delta(S) \ge \min_{1\le i\le L} \min_{p\in S'_i} d_i^*(p)$, proving the first equality.

It remains to prove that we can restrict the value of i to $L-D, L-D+1, \ldots, L$. We know from Lemma 2.3(1) that $\min_{p\in S'_i} d_i^*(p) > \delta_i/4D$. Moreover, we know from Lemma 2.1(2) that for $i < L-D$, $\delta_i/4D \ge \delta_{L-D-1}/4D \ge (2^{D+1}/4D)\cdot\delta_L \ge \delta_L \ge \delta(S)$. □

Now we are ready to describe how to find the closest pair using the sparse partition. According to the characterization of $\delta(S)$ in Lemma 2.4, we will augment the sparse partition with a data structure which stores, for each level $i \in \{1, \ldots, L\}$, the set of restricted distances $\{d_i^*(p) : p \in S'_i\}$.

The data structure that we use for this purpose is a *heap*. (See, e.g., [6, Chapter 7].) A heap storing n items can be built in linear time, and the operations *insert*(item), *delete*(item), and *change_key*(item, key)—which changes the key of the item item to the value key—are supported in $O(\log n)$ time. Also, operation *find_min*(H) can be performed in $O(1)$ time, and *find_all_min*(H), which returns all A items with minimum key, can be performed in time $O(A)$.

So, for each $i \in \{1, \ldots, L\}$, we maintain a min-heap H_i which stores items having the restricted distances $\{d_i^*(p) : p \in S'_i\}$ as their keys. How we compute these values depends on the way we implement the sparse partition, which will be described in the following sections, where we also describe the exact contents of our heap items.

LEMMA 2.5. *Let $S'_1, S'_2, \ldots, S'_L$ be the sparse sets of a sparse partition for S, and for each $1 \le i \le L$, let the set $\{d_i^*(p) : p \in S'_i\}$ of restricted distances be stored*

in a min-heap H_i. Then the minimum distance $\delta(S)$ can be found in constant time. Moreover, all point pairs attaining this minimum distance can be reported in time proportional to their number.

Proof. For $i \leq 0$, define H_i as the empty heap. Lemma 2.4 characterizes $\delta(S)$ as a minimum of certain restricted distances. In particular, Lemma 2.4 says that $\delta(S)$ can only be stored in one of the heaps $H_{L-D}, H_{L-D+1}, \ldots, H_L$. To find $\delta(S)$ it is therefore enough to take the minima of these $D+1$ heaps and then to take the minimum of these $D+1$ values. Moreover, we can report all closest pairs in time proportional to their number as follows: in all of the at most $D+1$ heaps whose minimum key is $\delta(S)$, we report all items whose key is equal to $\delta(S)$. From the discussion of heaps above, this can be done in time proportional to the number of items that are reported. □

We close this section with an abstract description of our data structure.

The closest-pair data structure for set S.

- A data structure storing S uniformly as a sparse partition according to Definitions 2.1 and 2.2.
- The heaps H_1, H_2, ..., H_L, where H_i stores the set of restricted distances $d_i^*(p)$, cf. Definition 2.3, for all points p in the sparse set S_i'.

In the rest of the paper, we discuss two different ways to implement the data structure. First, we describe a grid-based implementation. Since this data structure is the most intuitive one, we describe the update algorithms for this structure. Then we define the other variant of the data structure. Concerning implementation details and update algorithms, we then only mention the changes that have to be made in comparison with the grid-based implementation in order to establish the results.

3. A grid-based implementation of the sparse partition. Let S be a set of n points in D-space. To give a concrete implementation of a sparse partition for S, we only have to define the set S_i', i.e., the subset of sparse points in S_i for each i.

3.1. The notion of neighborhood in grids. We start with some definitions. Let $\delta > 0$. We use $\mathcal{G}_\delta$ to denote the grid with mesh size δ and a lattice point at $(0, 0, \ldots, 0)$. Hypercubes of the grid are called *boxes*. More precisely, a box has the form

$$[i_1\delta : (i_1+1)\delta) \times [i_2\delta : (i_2+1)\delta) \times \cdots \times [i_k\delta : (i_k+1)\delta)$$

for integers $i_1, \ldots, i_k$. We call $(i_1, \ldots, i_k)$ the *index* of the box. Note that with this definition of a box as the product of half-open intervals, every point in $\mathbb{R}^D$ is contained in exactly one grid box. The neighborhood of a box b in the grid $\mathcal{G}_\delta$, denoted by $N(b)$, consists of b itself plus the collection of $3^D - 1$ boxes bordering on it. Let p be any point in $\mathbb{R}^D$ and let $b_\delta(p)$ denote the box of $\mathcal{G}_\delta$ that contains p. The *neighborhood of p in $\mathcal{G}_\delta$*, denoted by $N_\delta(p)$, is defined as the neighborhood of $b_\delta(p)$; i.e., $N_\delta(p) := N(b_\delta(p))$. Let V be a set of points in $\mathbb{R}^D$. The *neighborhood of p in $\mathcal{G}_\delta$ relative to V* is defined as

$$N_\delta(p, V) := N_\delta(p) \cap (V \setminus \{p\}).$$

We say that p *is sparse in $\mathcal{G}_\delta$ relative to V* if $N_\delta(p, V) = \emptyset$, i.e., if, besides p, there are no points of V in $N_\delta(p)$. In cases where V and δ are understood from the context we will simply say that p is *sparse.*

The following observations follow directly from the definitions.

LEMMA 3.1. *Let V be a set of points in $\mathbb{R}^D$, and let p and q be arbitrary points in $\mathbb{R}^D$.*

(N.1) *If $N_\delta(p, V) = \emptyset$, then $d(p, V) > \delta$.*

(N.2) *If $q \in N_\delta(p, V)$, then $d(p, q) \leq 2D\delta$.*

(N.3) $q \in N_\delta(p) \iff p \in N_\delta(q)$.

We are now in a position to define the sets S'_i precisely. For $i \geq 1$, let

$$S'_i := \{p \in S_i \,:\, p \text{ sparse in } \mathcal{G}_{\delta_i/4D} \text{ relative to } S_i\}. \tag{1}$$

For convenience, we now modify the abstract definition of the sparse partition given in Definition 2.1 and replace it by one which is defined in terms of grid neighborhoods. Recall that in Definition 2.1 S'_i was not fully specified; the definition only gave conditions, specifically (b.1), (b.2), (c), and (d), which S'_i had to satisfy. We now replace those conditions by equation (1); with this new definition the sparse partitions are totally specified and we will be able to use them in section 4 to develop and later analyze our update algorithms.

DEFINITION 3.1. *A sparse partition for the set S is a sequence of 5-tuples $(S_i, S'_i, p_i, q_i, \delta_i)$, $1 \leq i \leq L$, where L is a positive integer, such that*

1. *For $i = 1, \ldots, L$,*
   - (a) $\emptyset \neq S_i \subseteq S$;
   - (b) $p_i, q_i \in S_i$ *and* $p_i \neq q_i$ *if* $|S_i| > 1$;
   - (c) $\delta_i = d(p_i, q_i) = d(p_i, S_i)$;
   - (d) $S'_i = \{p \in S_i \,:\, N_{\delta_i/4D}(p, S_i) = \emptyset\}$.
2. $S_1 = S$ *and for* $1 \leq i \leq L-1$, $S_{i+1} = S_i \setminus S'_i$.

LEMMA 3.2. *Let $(S_i, S'_i, p_i, q_i, \delta_i)$, $1 \leq i \leq L$, be a set of 5-tuples satisfying Definition 3.1. Then this set of 5-tuples also satisfies Definition 2.1.*

Proof. We only have to prove conditions (b) and (c) of Definition 2.1. Let $1 \leq i \leq L$ and let $p \in S_i$. First assume that $p \notin S'_i$. Then there is a point $q \in S_i$ which is in the neighborhood of p. By (N.2), $d(p, S_i) \leq d(p, q) \leq 2D \cdot \delta_i/4D = \delta_i/2$. This proves condition (b.1). To prove (b.2), assume that $p \in S'_i$. Then, the neighborhood of p relative to S_i is empty. Hence, by (N.1), $d(p, S_i) > \delta_i/4D$.

To prove (c), let $1 \leq i < L$ and let $p \in S_{i+1} = S_i \setminus S'_i$. It follows that there is a point $q \in S_i$ such that $q \in N_{\delta_i/4D}(p)$. By the symmetry property (N.3), this is equivalent to $p \in N_{\delta_i/4D}(q)$ and therefore $q \in S_{i+1}$. From condition (b.1), we also have $d(p, q) \leq \delta_i/2$.

In Figure 1 we provide an example of a sparse partition based on Definition 3.1. □

We now come to some additional properties of the sparse partition as defined in Definition 3.1 that will be used in the dynamic maintenance of the data structure. For this purpose, we give some additional facts about neighborhoods.

We start with some notation. Let p be a point in $\mathbb{R}^D$. We number the 3^D boxes in the neighborhood of p as follows. The number of a box is a D-tuple over $\{-1, 0, 1\}$. The jth component of the D-tuple is -1, 0, or 1, depending on whether the jth coordinate of the box (i.e., its lower left coordinate) is smaller than, equal to, or greater than the corresponding coordinate of $b_\delta(p)$. We call this D-tuple the *signature* of a box. We denote by $b_\delta^\Psi(p)$ the box with signature Ψ in $N_\delta(p)$.

We now define the notion of *partial neighborhood* of a point p. (See Figure 2.) For any signature Ψ, we denote by $N_\delta^\Psi(p)$ the part of p's neighborhood that is in the neighborhood of $b_\delta^\Psi(p)$. Note that $N_\delta^\Psi(p)$ contains all the boxes $b_\delta^{\Psi'}(p)$ of $N_\delta(p)$ whose signature Ψ' differs from Ψ by at most 1 for each coordinate—these are exactly the

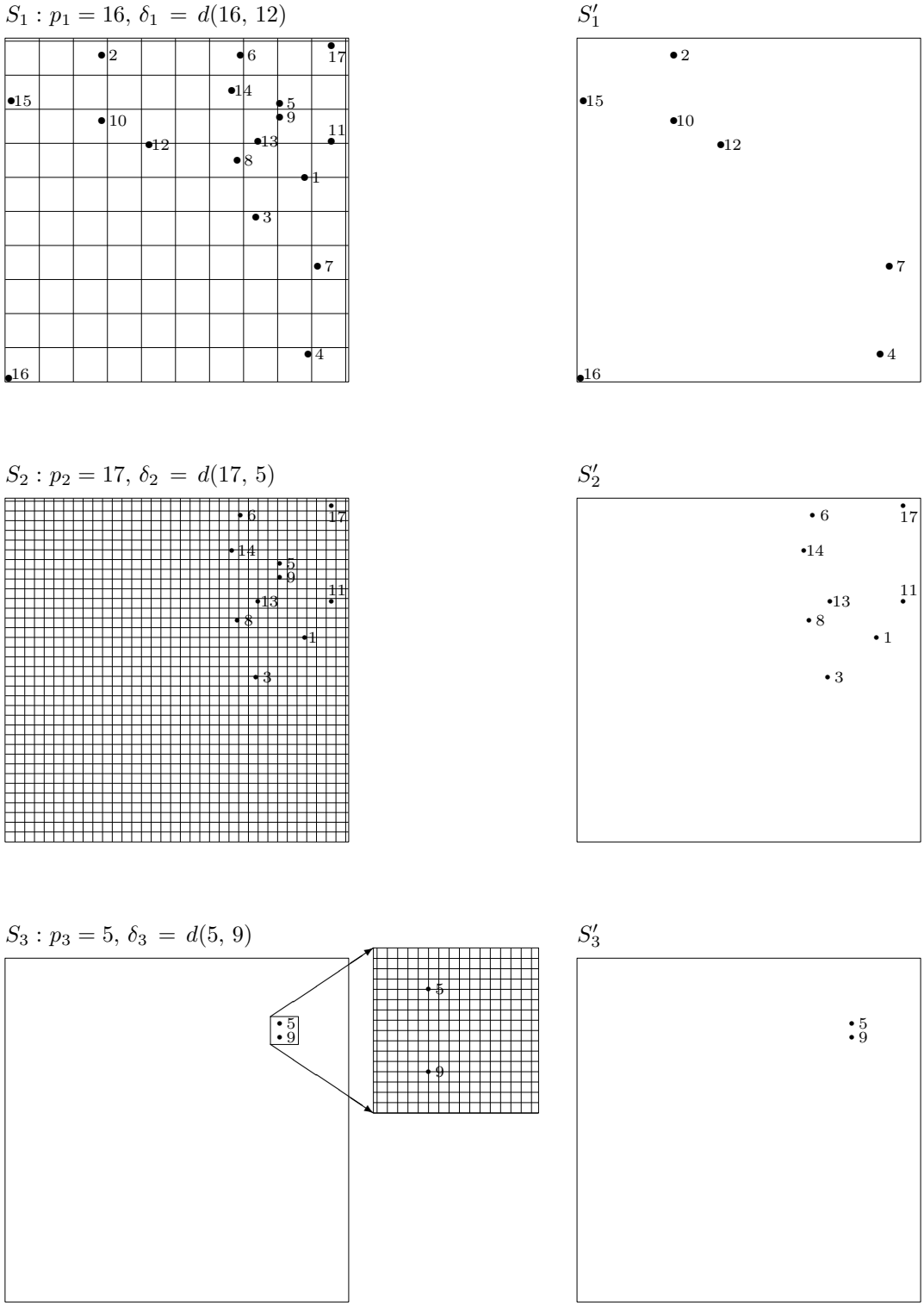


FIG. 1. *A sparse partition. Although the sets S'_i are also stored in grids, we have not shown the corresponding grids.*

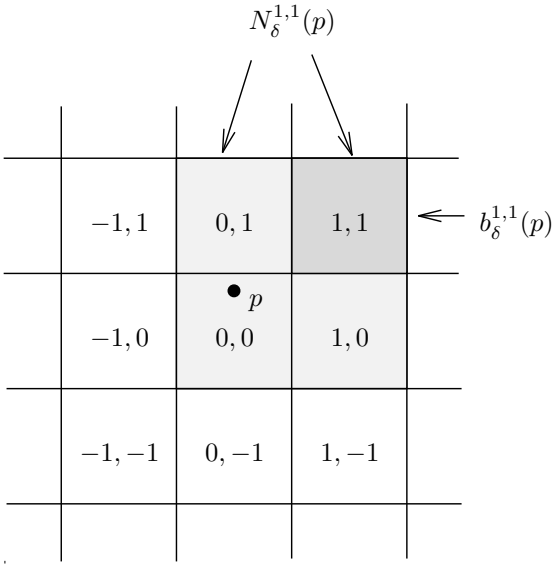


FIG. 2. *The neighborhood of a point p in $\mathcal{G}_\delta$. The dark shaded area denotes the box $b_\delta^{1,1}(p)$ in the upper right corner of p's neighborhood. This box also belongs to $N_\delta^{1,1}(p)$, the partial neighborhood of p with signature $1,1$. The light shaded area shows the other three boxes of $N_\delta^{1,1}(p)$.*

boxes bordering on $b_\delta^\Psi(p)$ including $b_\delta^\Psi(p)$ itself. In particular, $N_\delta^{0,\dots,0}(p) = N_\delta(p)$; i.e., the partial neighborhood with signature $0,\dots,0$ is the whole neighborhood of p.

The following properties will let us relate the neighborhoods of a point in different grids and more specifically in the different grids that correspond to different levels of the same sparse partition.

LEMMA 3.3. *Let $0 < \delta' \le \delta''/2$ be real numbers and let $p \in \mathbb{R}^D$. Then the following hold.*

(N.4) $N_{\delta'}(p) \subseteq N_{\delta''}(p)$.

(N.5) *For any signature $\Psi \in \{-1,0,1\}^D$: $b_{\delta'}^\Psi(p) \subseteq N_{\delta''}^\Psi(p)$.*

Proof. For any grid size δ and $1 \le j \le D$, denote by $h_\delta^{L,j}, h_\delta^{l,j}, h_\delta^{r,j}, h_\delta^{R,j}$ the jth coordinates of the four hyperplanes bounding the grid boxes of p's neighborhood in the j-direction, ordered from "left" to "right." See Figure 3.

Let $q = (q^{(1)}, q^{(2)}, \dots, q^{(D)}) \in \mathbb{R}^D$, and let $\Psi = (\alpha_1, \dots, \alpha_D) \in \{-1,0,1\}^D$ be a signature. Then $q \in b_\delta^\Psi(p)$ in $\mathcal{G}_\delta$ if and only if for all $1 \le j \le D$,

$$\begin{aligned} h_\delta^{l,j} \le q^{(j)} \le h_\delta^{r,j} &\qquad \text{if } \alpha_j = 0,\\ h_\delta^{r,j} \le q^{(j)} \le h_\delta^{R,j} &\qquad \text{if } \alpha_j = 1,\\ h_\delta^{L,j} \le q^{(j)} \le h_\delta^{l,j} &\qquad \text{if } \alpha_j = -1. \end{aligned}$$

Also, $q \in N_\delta^\Psi(p)$ if and only if for all $1 \le j \le D$,

$$\begin{aligned} h_\delta^{L,j} \le q^{(j)} \le h_\delta^{R,j} &\qquad \text{if } \alpha_j = 0,\\ h_\delta^{l,j} \le q^{(j)} \le h_\delta^{R,j} &\qquad \text{if } \alpha_j = 1,\\ h_\delta^{L,j} \le q^{(j)} \le h_\delta^{r,j} &\qquad \text{if } \alpha_j = -1. \end{aligned}$$

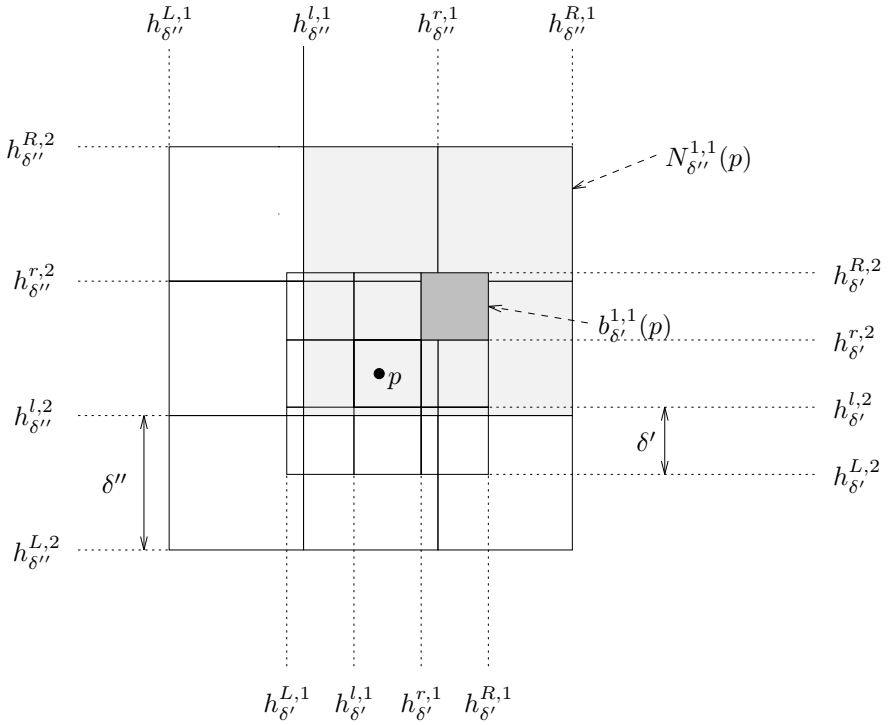


FIG. 3. *The neighborhoods of a point p in grids $\mathcal{G}_{\delta'}$ and $\mathcal{G}_{\delta''}$ where $\delta' \leq \delta''/2$.*

Figure 3 shows the neighborhoods of p in the two grids $\mathcal{G}_{\delta'}$ and $\mathcal{G}_{\delta''}$.

Now observe that, since $\delta' \leq \delta''/2$,

$$h^{L,j}_{\delta'} \geq h^{L,j}_{\delta''}, \tag{2}$$

$$h^{R,j}_{\delta'} \leq h^{R,j}_{\delta''} \tag{3}$$

for $1 \leq j \leq D$. These facts are equivalent to $N_{\delta'}(p) \subseteq N_{\delta''}(p)$, which is claim (N.4).

Furthermore, by the definition of the hyperplanes w.r.t. p,

$$h^{r,j}_{\delta'} \geq h^{l,j}_{\delta''}, \tag{4}$$

$$h^{l,j}_{\delta'} \leq h^{r,j}_{\delta''} \tag{5}$$

for $1 \leq j \leq D$. This proves claim (N.5): $b^{\Psi}_{\delta'}(p) \subseteq N^{\Psi}_{\delta''}(p)$. □

NOTATION. Consider a set S which is stored in a set of 5-tuples $(S_i, S'_i, p_i, q_i, \delta_i)$, $1 \leq i \leq L$, according to Definition 3.1. Since we will only use grids $\mathcal{G}_{\delta_i/4D}$ for the data structures that store level i of the partition, we will use the abbreviations $\mathcal{G}_i := \mathcal{G}_{\delta_i/4D}$ and $N_i(p) := N_{\delta_i/4D}(p)$ from now on. We use the same convention for the neighborhood relative to a set.

COROLLARY 3.1. *Let p be an arbitrary point of $\mathbb{R}^D$, and let $(S_i, S'_i, p_i, q_i, \delta_i)$, $1 \leq i \leq L$, be a sparse partition. Then for any $1 \leq i < j \leq L$, $N_j(p) \subseteq N_i(p)$.*

Proof. Apply (N.4) from Lemma 3.3 with $\delta'' = \delta_i/4D, \delta' = \delta_j/4D$, noting that $\delta' \leq \delta''/2$ by Lemma 2.1. □

In particular, if $N_i(p, S_i) = \emptyset$ for a point $p \in \mathbb{R}^D$, i.e., if p is sparse in $\mathcal{G}_i$ relative to S_i, then since $S_{i+1} \subseteq S_i$, Corollary 3.1 implies $N_{i+1}(p, S_{i+1}) = \emptyset$, which means

that p is also sparse in $\mathcal{G}_{i+1}$ relative to S_{i+1}. This property will be crucial to our update algorithms.

The following lemma will also be useful later on.

LEMMA 3.4. *Let $(S_i, S_i', p_i, q_i, \delta_i)$, $1 \le i \le L$, be a sparse partition. Then for any $p \in S \setminus S_{i+1}, 1 \le i < L$,*

$$N_i(p, S) = \emptyset.$$

Proof. We use induction on i. If $i = 1$, then for any $p \in S \setminus S_2 = S_1'$, $N_1(p, S) = N_1(p, S_1) = \emptyset$ by definition. Now let $i > 1$ and assume that, for any $p \in S \setminus S_i$, we have $N_{i-1}(p, S) = \emptyset$.

1. If $p \in S \setminus S_i$, then $N_i(p, S) \subseteq N_{i-1}(p, S) = \emptyset$ by Corollary 3.1 and our induction hypothesis, respectively.
2. If $p \in S_i'$, then $N_i(p, S_i) = \emptyset$ by definition. It remains to show that $N_i(p, S \setminus S_i) = \emptyset$. This is true because if there were a point $q \in S \setminus S_i$ such that $q \in N_i(p)$, then by the symmetry property (N.3) we have $p \in N_i(q)$, contradicting $N_i(q, S) = \emptyset$, which was shown in item 1 above.

We have thus shown that $N_i(p, S) = \emptyset$ for any $p \in S \setminus S_{i+1} = (S \setminus S_i) \cup S_i'$. □

3.2. Storing a point set according to a grid. We now explain how to store the point sets involved in the sparse partition of our input set S. Let δ be a grid size and $V \subseteq S$ be a subset of S. We store each nonempty box in the grid $\mathcal{G}_\delta$ in a hash table, using the index of the box in $\mathcal{G}_\delta$ as a key. Associated with each box b, we store a list containing the points in $V \cap b$ in arbitrary order. We call this *storing V according to $\mathcal{G}_\delta$*, and the data structure itself is called the *box dictionary*.

The dynamic perfect hashing algorithm of [10] allows us to store a set of integer-valued keys in linear space such that the information stored at a given key can be accessed in $O(1)$ worst-case time and permits a key to be inserted or deleted in $O(1)$ expected time. This algorithm needs to know in advance the range of integers from which the keys to be inserted or deleted may be drawn, but this range can easily be computed from δ if we know a frame containing all the points in S.

If V is stored according to $\mathcal{G}_\delta$, then we can answer the question *"are any points of V in box b?"* in $O(1)$ worst-case time. Moreover, if the answer is yes, we can report all points in $V \cap b$ in time proportional to their number. By checking all boxes in the neighborhood of an arbitrary point q, we can check in $O(1)$ time if q is sparse in the grid $\mathcal{G}_\delta$ relative to V, and by doing this for each point in V we can, in linear time, find the subset $V' \subseteq V$ of sparse points in V.

3.3. The complete data structure. Recall that, when discussing a sparse partition, we use $\mathcal{G}_i$ as a short form for the grid of mesh size $\delta_i/4D$. Our data structure now consists of the following:

For each $1 \le i \le L$,

- the pivot $p_i \in S_i$, its nearest neighbor q_i in S_i and $\delta_i = d(p_i, q_i)$,
- S_i stored according to $\mathcal{G}_i$,
- S_i' stored according to $\mathcal{G}_i$,
- the heap H_i.

Note that this means that S_i and S_i' are kept in two separate grid data structures defined on $\mathcal{G}_i$. We now add some more details to the description. Let b be a box of $\mathcal{G}_i$ which is nonempty w.r.t. S_i or S_i'. The list of points in $S_i \cap b$ will be called $\mathcal{L}(b)$, and the list of points in $S_i' \cap b$ will be called $\mathcal{L}'(b)$. Each element of $\mathcal{L}(b)$ is a record containing the following information:

$p \in \mathcal{L}(b)$: record | *point:* p | *upper:*$\uparrow p$ in S_{i-1} | $lower : \uparrow p$ in S_{i+1} |

Here, "$\uparrow p$ in V" means a pointer to the representation of point p in the data structure storing V. The pointers are nil if the corresponding representation of the point does not exist.

Each element of $\mathcal{L}'(b)$ is a record with the following information:

$p \in \mathcal{L}'(b)$: record | *point:* p | $it : \uparrow it(p)$ in H_i | $left : \uparrow p$ in S_i |

Here, "$\uparrow it(p)$ in H_i" means a pointer to the heap item $it(p)$ with key $d_i^*(p)$. (See below.) Note that each list $\mathcal{L}'(b)$ normally contains at most one point, by the sparseness property of the set S_i', but may temporarily contain more than one point during an update.

Now let us turn to the heaps. The key of an item in heap H_i is the value $d_i^*(p)$ for some $p \in S_i'$. Let q—if it exists—be such that $d_i^*(p) = d(p,q) < \delta_i$, and let l be such that $0 \le l \le D$ and $q \in S_{i-l}'$. Then the heap item $it(p)$ of H_i contains the following information:

item $it(p) \in H_i$: record | *key:* $d_i^*(p)$ | *point:* $\uparrow p$ in S_i' | *point*2: $\uparrow q$ in S_{i-l}' |

If the point q does not exist, i.e., if $d_i^*(p) = \delta_i$, then the pointer *point*2 is nil.

We are now in a position to describe a complete procedure which constructs the sparse partition and its auxiliary data structures. It will be convenient to have two slight variants of the procedure. The first, called $Build(T, j)$, takes as arguments a set of points T and an integer j and stores set T uniformly in levels $j, j+1, \ldots$ of a sparse partition. The second, $Near\_Build(T, p, j)$, again stores the given set of points T uniformly in levels $j, j+1, \ldots$ of a sparse partition but uses the given point $p \in T$ as the pivot for level j. In all invocations of $Near\_Build$, p will be chosen at random from T.

ALGORITHM $Near\_Build(T, p, j)$

1. $i := j$; $S_i := T$; $p_i := p$.
2. Calculate $\delta_i := d(p_i, S_i)$. Let $q_i \in S_i$ be such that $d(p_i, q_i) = \delta_i$.
3. Store S_i according to $\mathcal{G}_i$.
4. Compute $S_i' := \{p \in S_i : p \text{ sparse in } \mathcal{G}_i \text{ relative to } S_i\}$.
5. Store S_i' according to $\mathcal{G}_i$.
6. Compute the restricted distances $\{d_i^*(p) : p \in S_i'\}$ and, using a linear time algorithm, construct a heap H_i containing these values with the minimal value at the top.
7. If $S_i = S_i'$ stop; otherwise set $S_{i+1} := S_i \setminus S_i'$; choose a random point $p_{i+1} \in S_{i+1}$; set $i := i+1$; and goto 2.

ALGORITHM $Build(T, j)$

Choose a random point $p \in T$ and call $Near\_Build(T, p, j)$.

For the sake of simplicity, we did not mention in this algorithm how to establish the above-described links between the various parts of the data structure. The links between heap items and points in a list $\mathcal{L}'(b)$, i.e., points stored in a sparse set S_i', can be installed during the construction of the heaps. The pointers between representations of a point p in subsequent nonsparse sets S_i, S_{i+1} can be established easily in step 7 when S_{i+1} is obtained by stripping off the sparse set S_i' from S_i.

LEMMA 3.5. *Let $(S_i, S_i', p_i, q_i, \delta_i), 1 \le i \le L$, be a sparse partition, and let $p \in S_i'$ for some $i \in \{1, \ldots, L\}$. If we have the data structures storing the sets S_j' according to $\mathcal{G}_j$ available for $1 \le j \le i$, then the value $d_i^*(p)$ can be computed in $O(1)$ time.*

Proof. We know from Lemma 2.3(2) that if $d_i^*(p) = d(p,q)$ with $d(p,q) < \delta_i$ then q must be in one of the sets $S'_i, S'_{i-1}, \ldots, S'_{i-D}$. Furthermore, there are only a constant number of boxes in the grids $\mathcal{G}_j$, $i - D \le j \le i$, in which the point q can possibly appear: since the boxes in the grids $\mathcal{G}_j$ have side length $\delta_j/4D \ge \delta_i/4D$, these are the grid boxes that are within $4D$ boxes of the box in which p is located. Finally, because of the sparseness of the sets S'_j, there can be at most one point in each grid box and using the hash tables storing S'_j, $i - D \le j \le i$, we can find all points contained in these boxes and compute $d_i^*(p)$ in $O(1)$ time. □

LEMMA 3.6. *Let T be a set of points in $\mathbb{R}^D$. The procedures $Build(T, j)$ and $Near\_Build(T, p, j)$ complete in $O(|T|)$ expected time and produce sparse partitions of $O(|T|)$ expected size.*

Proof. For $k = 0, 1, \ldots$, consider the kth iteration of algorithm $Build(T, j)$ for which the loop index i has value $j+k$. Step 2 can be performed in $O(|S_i|)$ deterministic time by calculating the distance between p_i and all other points in S_i. Steps 3 and 5 build the grid data structures for S_i and S'_i and take $O(|S_i|)$ and $O(|S'_i|)$ expected time, respectively. By the discussion at the end of subsection 3.2, step 4, which computes S'_i from S_i, can be performed in $O(|S_i|)$ deterministic time. This implicitly includes the work of step 7.

Since we have the data structures for S'_l, $j \le l \le j + k$, available in the kth iteration of the algorithm, we can apply Lemma 3.5 to conclude that computing the restricted distances $\{d_i^*(p) : p \in S'_i\}$ in step 6 takes $O(|S'_i|)$ worst-case time. The heap H_i can be constructed within the same time bound.

Therefore, the expected running time of the algorithm is bounded by $O(\mathrm{E}(\sum_i |S_i|))$, which is also the amount of space used. Lemma 2.2 shows that this quantity is $O(|T|)$.

The analysis for $Near\_Build$ is similar. □

Recall that given this data structure, we can find the closest pair in S in $O(1)$ time by Lemma 2.5.

4. Dynamic maintenance of the data structure. In this section, we show how to maintain the sparse partition when the input set S is modified by insertions and deletions of points. The algorithms for insertions and deletions turn out to be very similar. We will demonstrate the ideas that are common to both update operations when we treat insertions. Then we give the deletion algorithm.

4.1. The insertion algorithm. We start with an intuitive description of the insertion algorithm. Let S be the current set of points, and assume we want to insert the point q. Further assume that S is already uniformly stored in the sparse partition. Our goal is to randomly build a sparse partition which uniformly stores $S \cup \{q\}$.

If we were to build a uniform sparse partition of $S \cup \{q\}$ from scratch then q would be the pivot of S_1 with probability $1/(|S_1| + 1)$; otherwise one of the points in S is chosen at random as the pivot. By assumption, p_1 (the pivot of S_1) is a random element of $S_1 = S$. Therefore, to generate a pivot for $S_1 \cup \{q\}$ it suffices to retain p_1 as pivot with probability $|S_1|/(|S_1|+1)$ and to choose q with probability $1/(|S_1|+1)$. If q is chosen, then we discard everything and run $Near\_Build(S_1 \cup \{q\}, q, 1)$, terminating the procedure. The latter event happens, however, only with probability $1/(|S_1| + 1)$ and so its expected cost is $O(1)$.

Assume now that p_1 remains unchanged as the pivot. We now check to see if q_1—the nearest neighbor of p_1—and hence, δ_1, have to be changed. First note that q can be the nearest neighbor of at most $3^D - 1 \le 3^D$ points in S_1. (See [8].) This means that δ_1 changes only if p_1 is one of these points. Since the updates are assumed to be independent of the coin flips of the algorithm, and since p_1 is chosen uniformly from

S_1, it follows that the probability of δ_1 changing is at most $3^D/|S_1|$. If δ_1 changes, we run $Build(S_1 \cup \{q\}, 1)$ and terminate the procedure. The expected cost of this is $O(1)$. The previous two steps are called "check for rebuild" in the later part of this section.

Assume now that p_1, q_1, and δ_1 remain unchanged. Let us denote $S \cup \{q\}$ by $\widetilde{S}$. We now need to determine the set $\widetilde{S}_2$, which contains the nonsparse points in $\widetilde{S}_1 = \widetilde{S}$. If q is sparse in S_1, it will go into $\widetilde{S}'_1$, and nothing further needs to be done; that is, the tuples $(S_i, S'_i, p_i, q_i, \delta_i)$ and $(\widetilde{S}_i, \widetilde{S}'_i, \widetilde{p}_i, \widetilde{q}_i, \widetilde{\delta}_i)$ are identical for $2 \le i \le L$. So, in this case, we can terminate the procedure. Otherwise, $\widetilde{S}_2$ contains q and possibly some points from S'_1. The set of points which are deleted from S'_1 due to the insertion of q is called $down_1$. This completes the construction of the first 5-tuple. Two of the three cases that may occur while constructing the first 5-tuple are given in Figures 4 and 5. The algorithm for constructing the new 5-tuples for $\widetilde{S}_i, i > 1$ follows the same general idea. We now describe more formally how to construct the new 5-tuples for $\widetilde{S}_i, i \ge 1$ and extend the notion of the set $down_1$ from the first level to the other levels of the sparse partition.

Let $i \ge 1$ and take $down_0 := \emptyset$. The following invariant holds if the algorithm attempts to construct the 5-tuple for $\widetilde{S}_i$ without having performed a rebuilding yet.

INVARIANT $\mathsf{INS}(i)$

(a) For $1 \le j < i$,

(a.1) $q \in \widetilde{S}_j$ and the set of 5-tuples $(\widetilde{S}_j, \widetilde{S}'_j, \widetilde{p}_j, \widetilde{q}_j, \widetilde{\delta}_j)$ satisfies Definitions 3.1 (sparse partition) and 2.2 (uniformity), where $\widetilde{p}_j = p_j, \widetilde{q}_j = q_j, \widetilde{\delta}_j = \delta_j$;

(a.2) $\widetilde{S}_{j+1} = \widetilde{S}_j \setminus \widetilde{S}'_j$.

(b) The sets $down_j, 0 \le j < i$, have been computed and $\widetilde{S}_i = S_i \cup down_{i-1} \cup \{q\}$.

Note that at the beginning of the algorithm, $\mathsf{INS}(1)$ holds because $down_0 = \emptyset$. We will show later that 3^D is an upper bound on the size of the union of all the $down$ sets. Thus each single $down$ set has size at most 3^D.

Now let us construct the 5-tuple for $\widetilde{S}_i$. From invariant $\mathsf{INS}(i)$ (b), we have $\widetilde{S}_i = S_i \cup down_{i-1} \cup \{q\}$. As discussed above, to construct the first 5-tuple we had to take the new point q as new pivot with probability $1/(1+|S_1|)$. In general, constructing $(\widetilde{S}_i, \widetilde{S}'_i, \widetilde{p}_i, \widetilde{q}_i, \widetilde{\delta}_i)$ from $(S_i, S'_i, p_i, q_i, \delta_i)$ requires that up to $3^D + 1$ points (q as well as the points in $down_{i-1}$) be considered as new pivots and also increases the chance of one of these points being closer to the old pivot than the pivot's previous nearest neighbor. This would result in a rebuilding (see Figure 6), but as the probabilities only increase by a constant factor, the effect is negligible.

If no rebuilding takes place, we determine $\widetilde{S}'_i$, the set of sparse points in $\widetilde{S}_i$. We define the set $down_i$, which consists of the points of S that were already sparse at some level $j \le i$ but that will not be sparse at level i due to the insertion of q, as follows. Let $D_i := S'_i \cup down_{i-1}$. Then

$$down_i := N_i(q, D_i) = \{x \in D_i \,:\, x \in N_i(q)\}. \tag{6}$$

The set D_i is called the "candidate set" for $down_i$. We can compute $\widetilde{S}'_i$ as follows: throw out from D_i all elements that belong to $down_i$ and add q, if it is sparse in $\widetilde{S}_i$. (We shall prove later that the set $\widetilde{S}'_i$ computed in this way actually *is* the set of sparse points in $\widetilde{S}_i$.)

We have constructed the 5-tuple $(\widetilde{S}_i, \widetilde{S}'_i, \widetilde{p}_i, \widetilde{q}_i, \widetilde{\delta}_i)$ and can now compute $\widetilde{S}_{i+1} = \widetilde{S}_i \setminus \widetilde{S}'_i$, the next subset of our new sparse partition for $\widetilde{S}$. If $q \in \widetilde{S}'_i$ then, by the

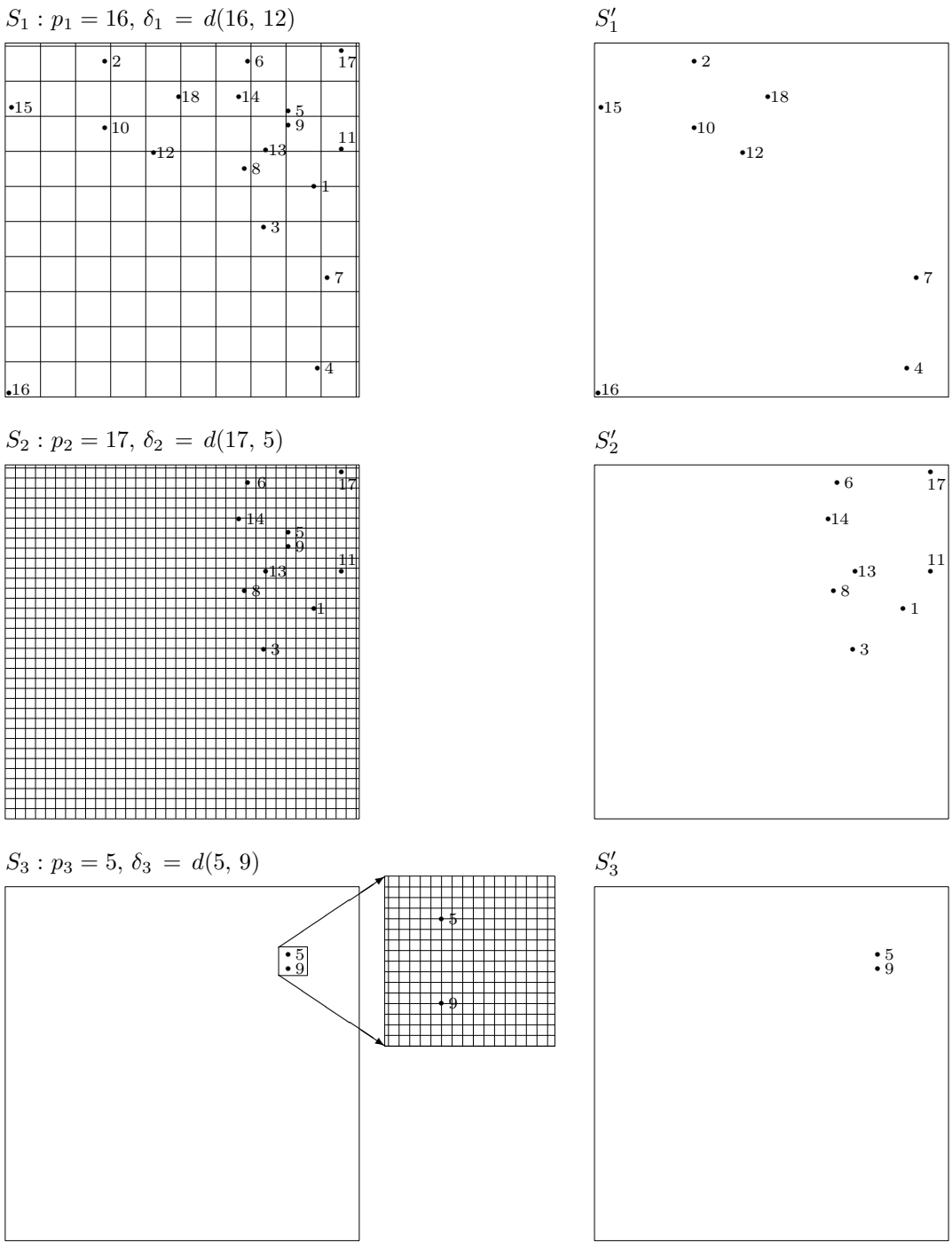


FIG. 4. *This is the sparse partition that resulted when point* 18 *was inserted into the sparse partition of the previous example. Point* 18 *was not chosen to be the pivot and was sparse in* S_1 *with the old pivot* $p_1 = 16$. *Thus* $18 \in S_1'$.

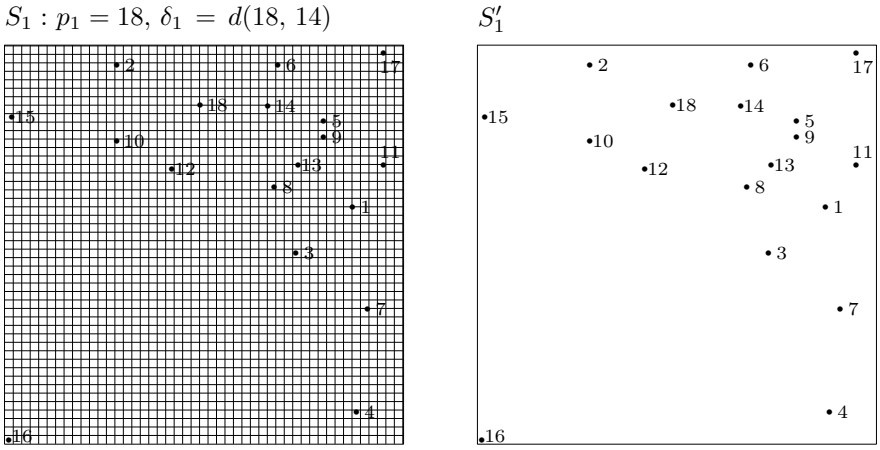


FIG. 5. *This is the sparse partition that results when the same point* 18 *is inserted as in the previous example but with point* 18 *now being chosen to be the pivot, something that occurs with probability* 1/18. *Note that in this case every point in* S_1 *becomes sparse so the partition only has one level.*

definition of the *down* sets, $down_i = \emptyset$ and $S_{i+1} = \widetilde{S}_{i+1}$. This means that the levels $i+1, \ldots, L$ of the sparse partition remain unchanged, and we are finished with the construction of the sparse partition for $\widetilde{S}$. Otherwise, $q \in \widetilde{S}_{i+1}$. So q and the points in $down_i$ are not sparse in $\widetilde{S}_i$ and we can add q and $down_i$ to S_{i+1}, giving the set $\widetilde{S}_{i+1}$. The invariant $\mathsf{INS}(i+1)$ holds, as we will prove later. We then continue with level $i+1$.

After the sparse partition has been updated, it remains to update the heaps. It is clear that the new point q has to be inserted into the heap structure appropriately. To see what kind of changes will be performed for the points of S, let us examine the point movements between the different levels of the sparse partition due to the insertion of q more closely. Let us look at level i, where $i \geq 1$. From invariant $\mathsf{INS}(i)$ (b) (respectively, $\mathsf{INS}(i+1)$ (b)), the points in $down_{i-1}$ (respectively, $down_i$) move at least down to level i (respectively, level $i+1$). The construction rule for $\widetilde{S}'_i$ now implies $\widetilde{S}'_i \setminus \{q\} = (S'_i \cup down_{i-1}) \setminus down_i$. Thus we have the following lemma.

LEMMA 4.1. *Let* p *be a point in* S*:*

(i) $p \in down_i \setminus down_{i-1} \iff p \in S'_i$ *and* $p \notin \widetilde{S}'_i$,
(ii) $p \in down_{i-1} \setminus down_i \iff p \notin S'_i$ *and* $p \in \widetilde{S}'_i$,
(iii) $p \in down_{i-1} \cap down_i \iff p \notin S'_i$ *and* $p \notin \widetilde{S}'_i$.

That is, the points in (i) *start moving* at level i, the points in (ii) *stop moving* at level i, and the points in (iii) *move through* level i. For all the points satisfying (i) or (ii), we have to update the heaps where values associated with these points disappear (i) or enter (ii).

The complete insertion algorithm is given in Figure 7.

It remains to describe the procedures that actually perform the heap updates. Before we do this, however, we will show that steps 1–3 in lines (3)–(21) of the algorithm actually produce a sparse partition for the new set $\widetilde{S}$.

LEMMA 4.2. *Assume algorithm Insert*(q) *has computed* 5*-tuples* $(\widetilde{S}_j, \widetilde{S}'_j, \widetilde{p}_j, \widetilde{q}_j, \widetilde{d}_j)$, $1 \leq j < i$, *and a set* $\widetilde{S}_i$ *that satisfy* $\mathsf{INS}(i)$. *Then if no rebuilding occurs, the* i*th*

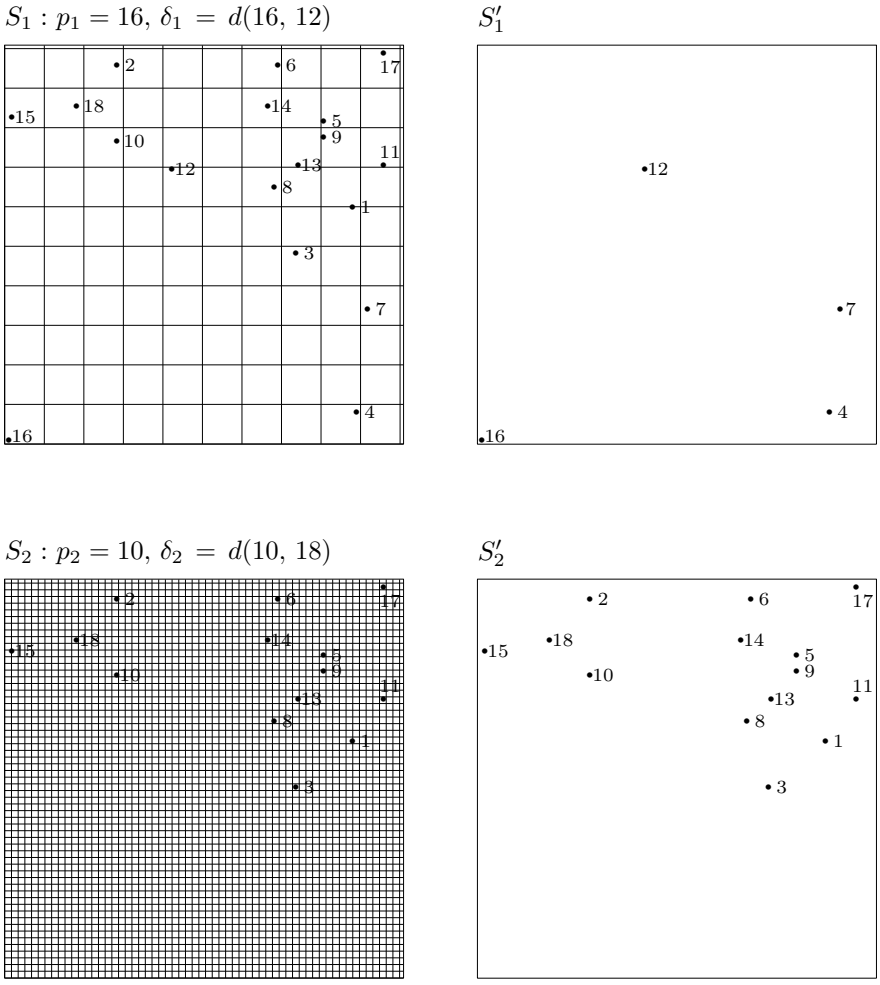


FIG. 6. *In this diagram a different point* 18 *is added to the first sparse partition. This point* 18 *is not sparse; it has neighbors* 2, 15, *and* 10. *In fact, because of the insertion of* 18, *these three points, which used to be sparse, are no longer sparse and are therefore placed in* $down_1$. *When the algorithm reached* S_2 *it then decided, probabilistically, that* 10 *should be the new pivot of that level. After making* 10 *the pivot, it found that all points were sparse and so terminated the algorithm.*

iteration of the algorithm constructs the 5*-tuple* $(\widetilde{S}_i, \widetilde{S}'_i, \widetilde{p}_i, \widetilde{q}_i, \widetilde{\delta}_i)$ *and the set* $\widetilde{S}_{i+1}$, *which satisfy* $\mathsf{INS}(i+1)$ (a.1)–(a.2). *Furthermore, if* $q \notin \widetilde{S}'_i$, *then* $\mathsf{INS}(i+1)$ (b) *also holds.*

Proof. Let us first prove (a.1), saying that the 5-tuple $(\widetilde{S}_i, \widetilde{S}'_i, \widetilde{p}_i, \widetilde{q}_i, \widetilde{\delta}_i)$ satisfies Definitions 3.1 and 2.2, with $\widetilde{p}_i = p_i, \widetilde{q}_i = q_i$, and $\widetilde{\delta}_i = \delta_i$. The 5-tuple is certainly uniform, and it retains the pivot as well as the pivot's nearest neighbor when the algorithm has passed step 2 (check for rebuild) of the algorithm without a rebuilding; cf. the discussion at the beginning of this section.

It remains to show that $N_i(p, \widetilde{S}_i) = \emptyset \iff p \in \widetilde{S}'_i$ for any $p \in \widetilde{S}_i$; see Definition 3.1, 1(d). We have $\widetilde{S}_i = S_{i+1} \cup S'_i \cup down_{i-1} \cup \{q\}$ from invariant $\mathsf{INS}(i)$ (b). Since $N_i(p, \widetilde{S}_i) \neq \emptyset$ for $p \in S_{i+1}$, it remains to prove the claim for $p \in D_i = S'_i \cup down_{i-1}$

(1) **Algorithm** Insert (q);
(2) **begin**
(3) **1. initialize:** $i := 1$; $down_0 := \emptyset$; $h := \infty$
(∗ From invariant $\mathsf{INS}(i)$ (b), we know that $\widetilde{S}_i = S_i \cup down_{i-1} \cup \{q\}$. We want to determine $\widetilde{S}'_i$. Before that, we check if the data structure has to be rebuilt. ∗)
2. check for rebuild:
(4) flip an $|\widetilde{S}_i|$-sided coin, giving pivot $\widetilde{p}_i$ of $\widetilde{S}_i$;
(5) **if** $\widetilde{p}_i \notin S_i$ **then**
(6) $Near\_Build(\widetilde{S}_i, \widetilde{p}_i, i)$; $h := i - 1$; goto **4.**
(7) **else**
(8) the old pivot p_i of S_i is also the pivot for $\widetilde{S}_i$
(9) **fi**;
(10) **if** $d(p_i, p) < \delta_i$ for some $p \in down_{i-1} \cup \{q\}$ **then**
(11) $Build(\widetilde{S}_i, i)$; $h := i - 1$; goto **4.**
(12) **else**
(13) do nothing (∗ $d_i = d(p_i, S_i) = d(p_i, \widetilde{S}_i)$ ∗)
(14) **fi**;
3. Determine $\widetilde{S}'_i$:
(15) compute the set $down_i$ from its candidate set $D_i = S'_i \cup down_{i-1}$; see equation (6);
(16) $\widetilde{S}'_i := D_i \setminus down_i$; $\widetilde{S}_{i+1} := S_{i+1} \cup down_i$; (∗ now $\widetilde{S}_{i+1} = (\widetilde{S}_i \setminus \widetilde{S}'_i) \setminus \{q\}$ ∗)
(17) **if** $N_i(q, \widetilde{S}_i) = \emptyset$ **then**
(18) add q to $\widetilde{S}'_i$; goto **4.** (∗ q is sparse in $\widetilde{S}_i$, and so $\widetilde{S}_{i+1} = S_{i+1}$ ∗)
(19) **fi**; (∗ q is not sparse in $\widetilde{S}_i$ ∗)
(20) add q to $\widetilde{S}_{i+1}$;
(21) $i := i + 1$; goto **2.**
4. Update heaps:
(∗ **Invariant:** $q \notin \widetilde{S}'_\ell$ for $\ell < i$.
Also $\min\{i, h\}$ is $h = i - 1$, if a rebuilding was performed.
Otherwise, $h = \infty$ and so $\min\{i, h\} = i$. ∗)
(22) **for** $\ell := 1$ **to** $\min\{i, h\}$ **do**
(23) **forall** $p \in down_\ell \setminus down_{\ell-1}$ **do**
(24) removefromheap(p, ℓ)
(25) **od**
(26) **forall** $p \in down_{\ell-1} \setminus down_\ell$ **do**
(27) addtoheap(p, ℓ)
(28) **od**;
(29) **od**;
(30) **if** $q \in \widetilde{S}'_{\min\{i,h\}}$ **then**
(31) addtoheap$(q, \min\{i, h\})$
(32) **fi**;
(33) **end**;

FIG. 7. *Algorithm Insert*(q). *The heap update procedures* addtoheap *and* removefromheap *called in step* 4 *will be given later.*

and $p = q$. Note that since $D_i \subseteq S \setminus S_{i+1}$ we have $N_i(p, S) = \emptyset$ by Lemma 3.4 and, therefore, for any $p \in D_i$,

$$\begin{aligned} N_i(p, \widetilde{S}_i) = \emptyset &\iff q \notin N_i(p) \\ &\iff p \notin N_i(q) && \text{by symmetry (N.3)} \\ &\iff p \notin \mathit{down}_i && \text{by definition of } \mathit{down}_i \\ &\iff p \in \widetilde{S}'_i && \text{by definition of } \widetilde{S}'_i. \end{aligned}$$

If $p = q$, then $N_i(p, \widetilde{S}_i) = \emptyset \iff q \in \widetilde{S}'_i$ by lines (17)–(19).

Next, we show that $\widetilde{S}_{i+1} = \widetilde{S}_i \setminus \widetilde{S}'_i$. After line (16), we have $\widetilde{S}_{i+1} = S_{i+1} \cup \mathit{down}_i$ and $\widetilde{S}'_i = (S'_i \cup \mathit{down}_{i-1}) \setminus \mathit{down}_i$, and at the end of step 3, q has been added to exactly one of these sets. Thus $\widetilde{S}_{i+1} \cup \widetilde{S}'_i = (S_{i+1} \cup S'_i) \cup \mathit{down}_{i-1} \cup \{q\} = S_i \cup \mathit{down}_{i-1} \cup \{q\}$ which equals $\widetilde{S}_i$ by $\mathsf{INS}(i)$ (b). Since $\widetilde{S}_{i+1}$ and $\widetilde{S}'_i$ are disjoint, it follows that $\widetilde{S}_{i+1} = \widetilde{S}_i \setminus \widetilde{S}'_i$.

Finally, if $q \notin \widetilde{S}'_i$, $\mathsf{INS}(i+1)$ (b) holds because $\widetilde{S}_{i+1} = S_{i+1} \cup \mathit{down}_i \cup \{q\}$ by lines (16) and (20). □

COROLLARY 4.1. *At the end of step* 3 *of algorithm Insert, we have computed a uniform sparse partition for* $\widetilde{S}$ *according to Definitions* 3.1 *and* 2.2.

Proof. Refer to Figure 7. Let $\hat{\imath}$ denote the value of i after the last completion of step 3. This in particular means that for each level $1 \le j < \hat{\imath}$, no rebuilding has yet occurred and $q \notin \widetilde{S}'_j$. By induction on the number of levels, $\mathsf{INS}(\hat{\imath})$ (a)–(b) hold. (We have already seen that $\mathsf{INS}(1)$ vacuously holds, forming the base of the induction. The induction step is established by Lemma 4.2.)

Invariant $\mathsf{INS}(\hat{\imath})$ (a) implies that the 5-tuples at levels $1, \ldots, \hat{\imath} - 1$ satisfy Definitions 3.1 and 2.2. Now the last iteration at level $\hat{\imath}$ is either a rebuilding or produces a 5-tuple such that $q \in \widetilde{S}'_{\hat{\imath}}$.

In the former case, the algorithm $\mathit{Build}(\widetilde{S}_{\hat{\imath}}, \hat{\imath})$ computes a uniform sparse partition for $\widetilde{S}_{\hat{\imath}}$, and the result is a uniform sparse partition for the set $\widetilde{S}$.

In the latter case, another application of Lemma 4.2 establishes $\mathsf{INS}(\hat{\imath}+1)$ (a). Let $\widetilde{L}$ denote the number of levels of the partition at the end of step 3. If $\widetilde{L} = \hat{\imath}$, then all the levels have been reconstructed and satisfy Definitions 3.1 and 2.2. Otherwise, if $\widetilde{L} > \hat{\imath}$, then some levels of the partition have not been reconstructed and thus $\widetilde{L} = L$. In this case, the 5-tuples for $\widetilde{S}_j$ are the old 5-tuples for S_j, $\hat{\imath} < j \le L$, which fulfill the desired property anyway. Therefore, all the 5-tuples $(\widetilde{S}_i, \widetilde{S}'_i, \widetilde{p}_i, \widetilde{q}_i, \widetilde{\delta}_i)$, $1 \le i \le \widetilde{L}$, are uniform and $\widetilde{S}_{i+1} = \widetilde{S}_i \setminus \widetilde{S}'_i$ for $1 \le i < \widetilde{L}$. It follows that this set of 5-tuples is a uniform sparse partition for $\widetilde{S}$. □

Having established the correctness of the algorithm, we now go into the implementation details in order to establish the running time. First, as promised, we examine the sizes of the *down* sets. The crucial fact which enables us to estimate the total size of the *down* sets is that at any level of the partition, only the new point q can make a previously sparse point nonsparse. We express this in the following lemma.

LEMMA 4.3. *Let the sets* $\mathit{down}_j, 1 \le j \le i$, *be defined, and let* $p \in \mathit{down}_j$ *for a level* $j \in \{1, \ldots, i\}$. *Then*

(1) $p \in N_j(q)$ *and*

(2) $N_j(p, S) = \emptyset$.

Proof. The first claim is obvious from the definition of down_j; cf. equation (6). The second claim is true because $p \in \mathit{down}_j$ implies $p \in S \setminus S_{j+1}$, and by Lemma 3.4, $N_j(p, S) = \emptyset$ for each $p \in S \setminus S_{j+1}$. □

LEMMA 4.4. *Let the sets $down_0, \ldots, down_i$ be as defined in equation* (6). *Then*

$$\Big| \bigcup_{1 \le j \le i} down_j \Big| \le 3^D.$$

Proof. Assume that $p \in down_j$ for some $j \le i$. Then $p \in N_j(q)$ and $N_j(p, S) = \emptyset$ by Lemma 4.3. Moreover, let $\Psi \in \{-1, 0, 1\}^D$ be such that $p \in b_j^\Psi(q)$. The partial neighborhood $N_j^\Psi(q)$ is the intersection of q's neighborhood with the neighborhood of p in the grid $\mathcal{G}_j$. Refer to Figures 2 and 3. Since $N_j(p, S) = \emptyset$, $N_j^\Psi(q)$ contains no point of $S \setminus \{p\}$.

Now consider a point $p' \in down_\ell$ for any $\ell > j$. From Lemma 4.3, we know that $p' \in N_\ell(q)$. Furthermore, assume that p' is in the box of q's neighborhood with signature Ψ; i.e., $p' \in b_\ell^\Psi(q)$. Since $\delta_\ell \le \delta_{j+1} \le \delta_j/2$ by Lemma 2.1(2), Lemma 3.3 (property (N.5)) gives $p' \in N_j^\Psi(q)$, from which it follows that p' must be identical to p.

This means that at levels $j + 1 \le \ell \le i$, there cannot be any point in $down_\ell$ with signature Ψ except p itself. (Note that a point can be in several *down* sets.) It follows that for each $\Psi \in \{-1, 0, 1\}^D$, there is at most one point p in S which satisfies $p \in down_j \wedge p \in b_j^\Psi(q)$. □

Computing the *down* sets in constant time. We have just shown that the total size of the *down* sets is constant, implying in particular that each single *down* set has constant size. Now we show that, given the candidate set $D_i = S_i' \cup down_{i-1}$, where S_i' is stored according to grid $\mathcal{G}_i$, we can compute $down_i$ in constant time. According to equation (6), we want to find all $p \in S_i' \cup down_{i-1}$ such that $p \in N_i(q, S_i' \cup down_{i-1})$. How do we find these points? The elements in S_i' are already stored at that level, whereas the elements in $down_{i-1} \cup \{q\}$ are not. We tentatively insert these points into the data structure storing the sparse set S_i' and then search in the neighborhood of q. This proves that we can find $down_i$ in constant time.

Performing the changes in the data structures storing the sparse partition. Of course, the changes from $(S_i, S_i', p_i, q_i, \delta_i)$ to $(\widetilde{S}_i, \widetilde{S}_i', \widetilde{p}_i, \widetilde{q}_i, \widetilde{\delta}_i)$ also have to be performed in the data structures that actually store the 5-tuple. We will now fill in these details.

The operations that we have to take care of are computing the new sparse set $\widetilde{S}_i'$ in lines (16) and (18) and computing the new set $\widetilde{S}_{i+1}$ in lines (16) and (20) of algorithm Insert.

To compute $\widetilde{S}_i' = (S_i' \cup down_{i-1}) \setminus down_i$, we just have to insert and delete a constant number of points in the data structure storing the sparse set S_i'. To insert a point p into $\widetilde{S}_i'$ (respectively, $\widetilde{S}_{i+1}$), we add p to the list $\mathcal{L}'(b)$ (respectively, $\mathcal{L}(b)$) where b is the box containing p. We also have to insert the box b into the box dictionary of the grid data structure, if it was not there before. This takes $O(1)$ expected time. (The same holds for the deletion of a box from the box dictionary.)

Now let us turn to the deletion of points. Note that during the insertion algorithm, deletions are performed in the sparse sets S_i'; more specifically there may be points that are in S_i' but are not in $\widetilde{S}_i'$. We can easily delete those points because we know that the lists $\mathcal{L}'(b)$ can only contain a constant number of points: at most one point at the start of the operation by the sparseness property, plus the points in $down_{i-1}$ that might have been tentatively inserted into the list. We remark here that instead of actually deleting the points of $down_i$ from the data structure storing the sparse set, we only *mark* them as deleted. The reason for this is that in step 4 when we

update the heaps we need to access both the old set S_i' and the new set $\widetilde{S}_i'$. The actual deletions will be performed after step 4 has been completed.

The lists $\mathcal{L}(b)$ for the nonsparse set S_{i+1} can contain more than a constant number of points. However, observe that a point is only deleted from a nonsparse set S_{i+1} during the insertion algorithm if a rebuilding occurs.

To sum up, performing the changes from the old 5-tuple $(S_i, S_i', p_i, q_i, \delta_i)$ to the new 5-tuple $(\widetilde{S}_i, \widetilde{S}_i', \widetilde{p}_i, \widetilde{q}_i, \widetilde{\delta}_i)$ of the sparse partition takes $O(1 + |down_{i-1}| + |down_i|)$ expected time.

LEMMA 4.5. *Steps* 1–3 *of algorithm Insert*(q) *take expected time* $O(\log n)$.

Proof. Consider one iteration of the steps 2 and 3. If no rebuilding occurs, the running time of step 2 is constant. (Recall that we assume that we can obtain a random number of $O(\log n)$ bits in constant time.) By the discussion in the two paragraphs before the lemma, the expected running time of step 3 at level i is $O(1 + |down_{i-1}| + |down_i|) = O(1)$.

We now give a probabilistic analysis for the insertion time, taking rebuildings into account. We show that the expected running time over all iterations of steps 1–3 is $O(\log n)$. The expectation is taken both over the new random choices and over the expected state of the old data structure.

Let the initial set of tuples be $(S_i, S_i', p_i, q_i, \delta_i)$, $1 \le i \le n$, padding the sequence out with empty tuples if necessary. Let T_i be the time to construct $\widetilde{S}_i$ from S_i assuming no rebuilding has taken place while constructing $\widetilde{S}_1, \ldots, \widetilde{S}_{i-1}$. Clearly, the overall running time X satisfies $X \le \sum_{i=1}^{n} T_i$. For $1 \le i \le n$, we have the following: with probability at most $\min(1, c/|S_i|)$ for some constant c, a rebuilding happens at level i and, therefore, $\mathrm{E}(T_i) = O(|S_i|)$ in this case. Otherwise, $\mathrm{E}(T_i) = O(1)$. These expected time bounds stem from the running times of the hashing algorithms that are used to rebuild or to update the structure, respectively. Since the random choices made by the hashing algorithms are independent of the coin flips made by our algorithms Insert and Build to choose the pivots, we can multiply the probabilities of the events with the expected running times that are valid for the event and so obtain that the expected value of T_i is $O(1)$, independently of the previous state of the data structure.

We now note that $\mathrm{E}(\sum_{i=k}^{n} T_i) = O(|S_k|)$ for any $1 \le k \le n$. This is because either a rebuild occurs at level k, requiring $O(|S_k|)$ expected time, and we are done. Otherwise $\mathrm{E}(T_k) = O(1)$ and by induction, $\mathrm{E}(\sum_{i=k+1}^{n} T_i) = O(|S_{k+1}|)$. Since $|S_{k+1}| \le |S_k| - 1$ (the pivot at level N is always sparse) we can choose the constant within the Big-Oh large enough to conclude that $\mathrm{E}(\sum_{i=k}^{n} T_i) = O(|S_k|)$ in this case as well.

Let $N = \lceil \log n \rceil$. From the above discussion,

$$\begin{aligned}\mathrm{E}(X) &\le \sum_{i=1}^{N} \mathrm{E}(T_i) + \mathrm{E}\left(\sum_{i=N+1}^{n} T_i\right) \\ &\le O(\log n) + O(\mathrm{E}(|S_{N+1}|)) = O(\log n)\end{aligned}$$

since $\mathrm{E}(|S_{\lceil \log n \rceil + 1}|)$ is $O(1)$. □

Remark. Note that not only is the running time at each level of the sparse partition a random variable, but also the number of levels is a random variable, and its value can be as high as n. This means in particular that the running times of consecutive update operations are not independent.

Discussion of the heap updates. We are now ready to discuss step 4 of the insertion algorithm. Heap updates are necessary when points move to a different level due to the insertion of q.

```
(1)   proc removefromheap(p, h);
(2)   begin
(3)       (* p starts moving at level i, i.e., p ∈ S'_i, but p ∉ S̃'_i *)
(4)       DELETE(H_i, it(p));
(5)       for ℓ := i to min{i + D, h} do
(6)           forall r ∈ S'_ℓ ∩ S̃'_ℓ such that d(r, p) < δ_ℓ do
(7)               CHANGE_KEY(it(r), d*_ℓ(r))
(8)           od
(9)       od;
(10)  end;
```

FIG. 8. *Procedure removefromheap*(p, h).

Assume point p moves to a different level. Then heap updates are necessary (i) when p starts moving at level i and (ii) when p stops moving at some level j, where $i < j$. In the first case, we basically perform a deletion of the heap values associated with p, while in the second case we perform the corresponding reinsertions into the heap structure. Note that the latter case does not occur if the data structure has been rebuilt at some level $i < l \leq j$. In this case, the rebuilding algorithm inserts the values associated with p into the heap structure.

Note that at each level i a point can be associated with only a constant number of heap values, which are located in the heaps $H_\ell, i \leq \ell \leq i + D$. From Lemma 2.3(3), we know that $d_i^*(p) = \min\left(\delta_i, d(p, S'_{i-D} \cup \cdots \cup S'_i)\right)$. Thus a point $p \in S'_i$ can be associated with a heap in two different ways: either there is a value $d_i^*(p)$ in H_i or for each $i \leq \ell \leq i + D$ there may be points $r \in S'_\ell$ such that $d(r, p)$ gives rise to $d_\ell^*(r)$ in H_ℓ.

Recall that $\widetilde{L}$ denotes the last level of the sparse partition *after* the update. In our heap update procedures given below, we want to rearrange the heaps such that heap $H_j, 1 \leq j \leq \widetilde{L}$, contains the values

$$\left\{ d_j^*(p) = \min\left(\widetilde{\delta}_j, d(p, \widetilde{S}'_{j-D} \cup \cdots \cup \widetilde{S}'_j)\right) \ : \ p \in \widetilde{S}'_j \right\}.$$

At the moment, the heaps contain the restricted distances w.r.t. the old sparse partition, except for the levels that have been rebuilt. We therefore take care that we only rearrange heaps at levels that have not been rebuilt. In step 4, a parameter h occurs. It denotes the last level that has not been rebuilt if such a rebuilding has taken place. Otherwise, $h = \infty$. The heap update procedures are shown in Figures 8 and 9.

LEMMA 4.6. *After step* 4 *of algorithm Insert*(q)*, the heap* H_i *stores the set* $\{d_i^*(p) \ : \ p \in \widetilde{S}'_i\}$ *for all levels* $1 \leq i \leq \widetilde{L}$*. Also, the running time of the procedures* addtoheap *and* removefromheap *is* $O(1)$ *plus the time spent on the heap operations. The number of heap operations that are performed in step* 4 *is constant.*

Proof. Recall that at the beginning of step 4 the new sparse partition is computed, and since the elements of S'_i that are not in $\widetilde{S}'_i$ have only been marked deleted, we have both sparse sets at hand at each level.

Notation. For each level i, we call points that remain sparse, i.e., the points in $S'_i \cap \widetilde{S}'_i$, the *passive* points and the points that cease or start being sparse, i.e., the points in $(S'_i \setminus \widetilde{S}'_i) \cup (\widetilde{S}'_i \setminus S'_i)$, the *active* points.

Claim. Exactly those restricted distances which can change due to the change of the sparse partition and which have not been handled by rebuilding have been treated by the procedures shown in Figures 8 and 9.

```
(1)   proc addtoheap(p, h);
(2)   begin
(3)       (* p stops moving at level j, i.e., p ∉ S'_j, but p ∈ S̃'_j *)
(4)       compute d*_j(p); let r be such that d*_j(p) = d(r, p) if it exists;
(5)       it(p) := new item; it(p).key := d*_j(p); it(p).point := p; it(p).point2 := r;
(6)       INSERT(H_j, it(p));
(7)       for ℓ := j to min{j + D, h} do
(8)           forall r ∈ S'_ℓ ∩ S̃'_ℓ such that d(r, p) < δ_ℓ do
(9)               CHANGE_KEY(it(r), d*_ℓ(r))
(10)          od
(11)      od;
(12)  end;
```

FIG. 9. *Procedure addtoheap*(p, h).

Proof of claim. At the beginning of step 4, we know h, the index of the last level for which heap H_h has to be reconstructed, if a rebuilding has taken place. In this case, the data structure has been rebuilt at level $h+1$. (Otherwise $h = \infty$.) We can therefore guarantee that our heap update procedures do not treat levels whose heaps have already been correctly computed by a rebuilding.

Now consider the levels where the heaps have to be rearranged. Heap H_i contains the restricted distances of points $p \in S'_i$ to points in $S'_{i-D} \cup \cdots \cup S'_i$. For the active points, the claim is clear: these are the points in the symmetric difference of S'_i and $\widetilde{S}'_i$. By Lemma 4.1, these are exactly the points in the symmetric difference of $down_{i-1}$ and $down_i$. Our heap update procedures are called exactly for these points and the restricted distances of these points are deleted in line (4) of removefromheap and inserted in line (6) of addtoheap, respectively.

For a passive point p, we only have to examine, at levels $j = i, \ldots, i-D$, the points that are

1. active at level j and
2. closer to p than the threshold distance δ_i.

These are exactly the points that are treated in lines (5)–(9) of removefromheap and in lines (7)–(11) of addtoheap.

Also note that for every point that is treated by the heap update procedures, either the corresponding heap item is deleted, if it belongs to the set of points that ceases being sparse at that level, or its restricted distance according to the new sparse partition is computed (and inserted if it is a point that starts being sparse). This establishes the correctness of the heap update procedures and step 4 of algorithm Insert.

Now let us look at the running time of the heap update procedures. Each restricted distance can be computed in $O(1)$ time by Lemma 3.5. Moreover, from the proof of Lemma 3.5 we know that the restricted distances can be computed by searching the area of at most $4D$ boxes away from p in the grids that store the sparse sets $S'_{i+l}, 0 \le l \le D$. Outside this area, the restricted distance of a point r cannot be affected by removal or insertion of p. Since we assume that the dimension D is fixed, the total number of heap operations carried out by the procedure is constant, and the time spent by the procedure not counting the heap operations is also constant. □

LEMMA 4.7. *Algorithm Insert*(q) *correctly maintains the data structure and takes expected time* $O(\log n)$.

Proof. From Lemma 4.2, steps 1–3 establish that $S \cup \{q\}$ is stored uniformly as a sparse partition. Also, from Lemma 4.6, the heaps are maintained correctly by step 4. This proves the correctness of the algorithm.

As shown in Lemma 4.5, steps 1–3 have expected cost $O(\log n)$. Now consider step 4. From Lemma 4.4 we know that the heap update procedures are only called for a constant number of points. Since by Lemma 4.6 one procedure call only performs $O(1)$ heap operations and, apart from these operations, performs only $O(1)$ additional work, the total time for step 4 is $O(\log n)$. ∎

4.2. The deletion algorithm. Now we come to the algorithm that deletes a point q from the data structure. Let $\widetilde{S}$ denote $S \setminus \{q\}$. Deletion is basically the reverse of insertion and may involve some points becoming sparse at their current levels due to the deletion of q, thus causing them to move up a few levels. In particular, points which move to lower levels during an insertion of q move back to their old levels when q is deleted directly afterward (provided no rebuilding takes place).

An insertion ends at the level where the new point q is sparse. Therefore, assuming that $q \in S'_\ell$, we have to delete q from S'_ℓ and also from all the sets S_i, $1 \le i \le \ell$. Note that in order to be able to delete q efficiently from the nonsparse sets S_i containing it, we linked the occurrence of a point in S_i to its occurrence in S_{i-1} and vice versa, if the corresponding level exists.

Although it looks natural to implement a deletion starting at the level ℓ where q is sparse and then walking up the levels, it is much easier to implement the deletion algorithm in a top-down fashion, as in the insertion algorithm. In the insertion algorithm, we collected in $down_i$ the points that were sparse at some level $j \le i$ but that were no longer sparse at level i due to the insertion of q. Now we want to collect in up_i the points that are nonsparse at level i but will be sparse there after a deletion.

The deletion algorithm starts at the top level and moves downward, as algorithm Insert. We define $up_0 := \emptyset$. Let $i \ge 1$. The following invariant, which is analogous to invariant $\mathsf{INS}(i)$ in algorithm Insert, holds if the algorithm attempts to construct the 5-tuple for $\widetilde{S}_i$ without having performed a rebuilding yet.

Invariant $\mathsf{DEL}(i)$

(a) This invariant is identical to $\mathsf{INS}(i)$, saying that the new 5-tuples at the levels $1, \ldots, i-1$ satisfy Definitions 3.1 and 2.2.

(b) The sets $up_j, 0 \le j < i$, have been computed and $\widetilde{S}_i = (S_i \setminus up_{i-1}) \setminus \{q\}$.

Note that at the start of the algorithm, $\mathsf{DEL}(1)$ holds because $up_0 = \emptyset$.

To construct the 5-tuple for $\widetilde{S}_i$, the deletion algorithm first checks if a rebuilding has to be performed, as does the insertion algorithm. Having done that, it constructs the new sparse set $\widetilde{S}'_i$ and, along with it, the nonsparse set $\widetilde{S}_{i+1}$. $\widetilde{S}'_i$ is computed from the previous sparse set S'_i by adding the points of up_i and deleting the points of up_{i-1}. Also, we obtain $\widetilde{S}_{i+1}$ by deleting the points of up_i from S_{i+1}. Now, q is still in $\widetilde{S}'_i$ or $\widetilde{S}_{i+1}$, depending on whether it was previously in S'_i or S_{i+1}, respectively. Deleting q from the set containing it finishes the computation of $\widetilde{S}'_i$ and $\widetilde{S}_{i+1}$. If q was sparse at level i, then $S_{i+1} = \widetilde{S}_{i+1}$ and the construction of the new sparse partition is complete. (Note that up_i must be empty in this case.) Otherwise, we go into the next iteration and construct the 5-tuple for $\widetilde{S}_{i+1}$.

When the new sparse partition is computed, the heaps have to be updated. Analogously to the insertion algorithm, (i) $p \in up_{i-1} \setminus up_i$ means p *starts moving* at level i, i.e., $p \in S'_i$ and $p \notin \widetilde{S}'_i$, (ii) $p \in up_i \setminus up_{i-1}$ means p *stops moving* at level i, i.e., $p \notin S'_i$ and $p \in \widetilde{S}'_i$, and (iii) $p \in up_{i-1} \cap up_i$ means that p *moves through* level i, i.e., $p \notin S'_i$

(1) **Algorithm** Delete (q);
(2) **begin**
(3) **1. initialize:** $i := 1$; $up_0 := \emptyset$; $h := \infty$
($*$ From invariant $\mathsf{DEL}(i)$ (b), we know that $\widetilde{S}_i = (S_i \setminus up_{i-1}) \setminus \{q\}$. $*$)
2. check for rebuild:
(4) ($*$ we do not need to flip a coin for a new pivot $*$)
(5) **if** q or an element of up_{i-1} is the pivot p_i or the nearest neighbor of p_i
(6) **then** $Build(\widetilde{S}_i, i)$; $h := i - 1$; goto **4.**
(7) **fi**; ($*$ $d_i = d(p_i, S_i) = d(p_i, \widetilde{S}_i)$ $*$)
3. Determine $\widetilde{S}'_i$:
(8) compute $up_i = \left\{p \in S_i \ : \ N_i(p, S_i) = \{q\}\right\}$;
(9) $\widetilde{S}'_i := (S'_i \cup up_i) \setminus up_{i-1}$; $\widetilde{S}_{i+1} := S_{i+1} \setminus up_i$; ($*$ now $\widetilde{S}_{i+1} \cup \widetilde{S}'_i = \widetilde{S}_i \cup \{q\}$ $*$)
(10) **if** $q \in \widetilde{S}'_i$ **then**
(11) delete q from $\widetilde{S}'_i$; goto **4.** ($*$ q is sparse in $\widetilde{S}_i$, and so $\widetilde{S}_{i+1} = S_{i+1}$ $*$)
(12) **fi**; ($*$ q is not sparse in $\widetilde{S}_i$ $*$)
(13) delete q from $\widetilde{S}_{i+1}$;
(14) $i := i + 1$; goto **2.**
4. Update heaps:
Completely analogous to algorithm Insert. At levels $1 \le \ell \le \min\{i, h\}$, we execute the heap update procedures for the points in the symmetric difference of up_{i-1} and up_i, and for the deleted point q, if we are on a level where q contributes a heap value.
(15) **end**;

FIG. 10. *Algorithm Delete*(q).

and $p \notin \widetilde{S}'_i$. As before, the points that start or stop moving cause heap updates. The deletion algorithm is described in Figure 10.

From the similarity of invariants $\mathsf{INS}(i)$(b) and $\mathsf{DEL}(i)$(b), it is easy to see that the arguments used in Lemma 4.4 to derive the bound on the size of the *down* sets carry over to the *up* sets, and thus we obtain $|\bigcup_{1 \le i \le L} up_i| \le 3^D$.

The computation of the *up* sets is slightly different from the computation of the *down* sets. In order to compute the *down* sets efficiently, we gave an alternative but equivalent definition for these sets (equation (6)) and showed that the alternative definition could be efficiently realized. The difficulty there was that the points in $down_i$ could come from the set $S \setminus S_{i+1} = S'_1 \cup \cdots \cup S'_i$, and there seemed to be no direct way of extracting the points of $down_i$ from this set. In contrast to the insertion case, the points that are nonsparse at level i but will be sparse there after a deletion are all contained in S_i. We can compute the set up_i in constant time as follows. From $\mathsf{DEL}(i)$(b), it follows that $p \in up_i$ if and only if $N_i(p, S_i) = \{q\}$. Checking this condition means finding all points in S_i having only q in their neighborhood. Using the symmetry property (N.3), this can be done in $O(1)$ time.

LEMMA 4.8. *Algorithm Delete*(q) *correctly maintains the data structure and takes expected time* $O(\log n)$.

Proof. The proofs of correctness and running time are analogous to those for the insertion algorithm and are therefore omitted. □

We summarize the results of this section in the following theorem:

THEOREM 4.1. *There exists a data structure which stores a set S of n points in $\mathbf{R}^D$ such that the minimal distance $\delta(S)$ can be found in $O(1)$ time, and all point pairs*

attaining $\delta(S)$ can be reported in time proportional to their number. The expected size of the structure is $O(n)$, and we can maintain the data structure as S is modified by insertions or deletions of arbitrary points in $O(\log n)$ expected time per update. The algorithms run on a RAM and use randomization. The bounds are obtained under the assumption that we know a frame that contains all the points that are in the set S at any time, that the floor function can be computed in constant time, and that the updates do not depend upon the random choices made by the data structure.

5. An algebraic computation tree implementation. The solution from the previous section uses a somewhat inelegant model, which is an uneasy marriage of the unit-cost RAM and the algebraic computation tree. This algorithm may also be a poor one to use in practice, as the integers (box indices) which are computed as intermediate results may be so large that they cannot be manipulated in constant time by the hashing routines. Even if we implement the box dictionary by search trees, dividing point coordinates by very small numbers (interpoint distances) may lead to numerical problems.

We now present an algorithm which fits into the algebraic computation tree model. It can be verified that the algorithm requires only addition, subtraction, comparison, and multiplication of real numbers to maintain the closest pair (although computing the actual value of the minimum distance $\delta(S)$ in the L_t-metric for $1 < t < \infty$ will require the tth root function as well). However, it is well known that the floor function used by the previous algorithm is very powerful: the maximum-gap problem requires $\Omega(n \log n)$ time in the algebraic computation tree model but can be solved in $O(n)$ time by adding this function. Hence, we may expect some increase in the time complexity of the operations, and this does indeed turn out to be the case.

Note that the floor function was only used to compute the grid box containing a given point. Therefore, we will modify the algorithm Theorem 4.1 by using a degraded grid for which we only need algebraic functions. The method we use already appears in [9] and [11]. We sketch the structure here and refer to these papers or [18] for details.

Consider a standard grid of mesh size δ. Fixing the origin as a lattice point, we divide the space into slabs of width δ in each dimension. Since we can identify a slab using the floor function, this gives rise to an *implicit* storage of the slabs. To avoid the use of the floor function, we store these slabs *explicitly* by keeping a dictionary for the coordinates of its endpoints in each dimension.

In contrast to the standard grid, a degraded grid is defined in terms of the point set stored in it. To emphasize this, we use the notation $\mathcal{DG}_{\delta,V}$ for a degraded δ-grid defined by the points of a set $V \subseteq \mathbf{R}^D$ in comparison with the grid $\mathcal{G}_\delta$. In a degraded δ-grid $\mathcal{DG}_{\delta,V}$, all boxes have sides of length at least δ, and the boxes that contain a point of V have sides of length at most 2δ. See Figure 11.

The degraded grid can be maintained under insertions and deletions of points in logarithmic time, and the box containing a point can be identified in logarithmic time as well.

In order to implement our data structure, we only have to define the sparse sets S'_i. The alignment of boxes in slabs enables us to transfer the notion of neighborhood directly from standard grids to degraded grids. The neighborhood of a box consists of the box itself plus the $3^D - 1$ boxes bordering on it.

Consider a degraded δ-grid $\mathcal{DG}_{\delta,V}$. As in the grid case, the neighborhood of a point $p \in \mathbf{R}^D$ is defined as the neighborhood of the box $b_{\delta,V}(p)$ that contains p; i.e., $N_{\delta,V}(p) := N(b_{\delta,V}(p))$. The notion of partial neighborhood is also defined analogously.

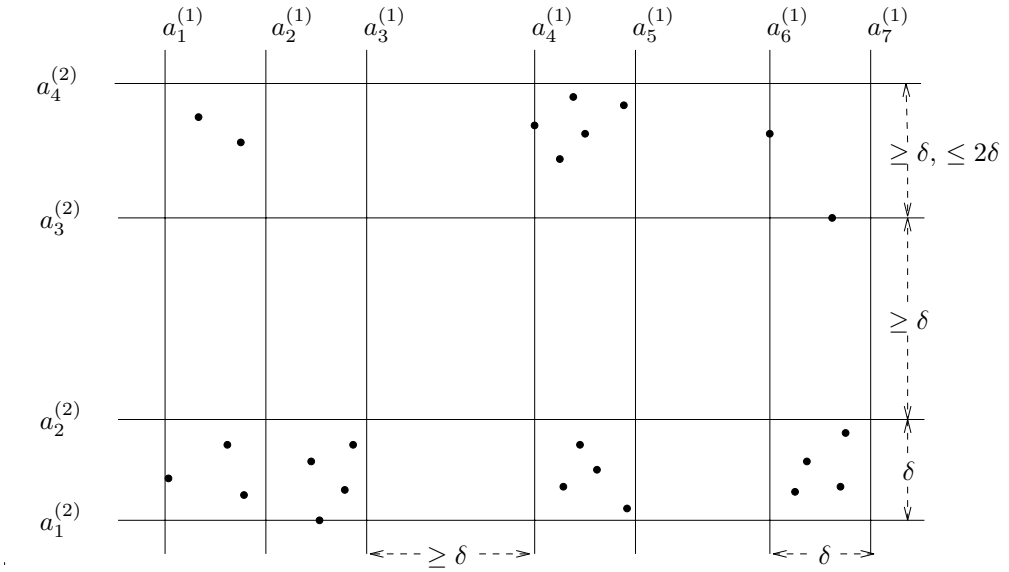


FIG. 11. *Example of a degraded δ-grid. It is dependent on the set stored in it.*

See Figure 2. We number the 3^D boxes in the neighborhood of a point p as described there, giving each box a *signature* $\Psi \in \{-1, 0, 1\}^D$. The box with signature Ψ is denoted by $b^{\Psi}_{\delta,V}(p)$. The boxes of p's neighborhood which are adjacent to $b^{\Psi}_{\delta,V}(p)$ form the *partial neighborhood of p with signature* Ψ, denoted by $N^{\Psi}_{\delta,V}(p)$.

We now define the neighborhood of a point relative to a set of points. For any set $\widehat{V}$, the neighborhood of p in $\mathcal{DG}_{\delta,V}$ relative to $\widehat{V}$, denoted by $N_{\delta,V}(p, \widehat{V})$, is defined as $N_{\delta,V}(p, \widehat{V}) := N_{\delta,V}(p) \cap (\widehat{V} \setminus \{p\})$. Note that in this definition the set $\widehat{V}$ need not be identical to the *defining set* V of the degraded grid. As before, we say that a point p is *sparse* in the degraded grid relative to $\widehat{V}$ if $N_{\delta,V}(p, \widehat{V}) = \emptyset$.

The basis of correctness and running time of the grid algorithms were the neighborhood properties (N.1)–(N.5). We now adapt these to handle degraded grids. Two changes are needed. First, we change the constants in response to the fact that a nonempty box might now have side length up to 2δ. Second, although we defined the neighborhood relative to a set $\widehat{V}$ independently of the defining set V of the degraded δ-grid, a lot of properties will only continue to hold if $\widehat{V} \subseteq V$. This is because boxes of a degraded δ-grid $\mathcal{DG}_{\delta,V}$ that do not contain a point of the defining set V may be unbounded. For example, this may cause a point $q \in \widehat{V}$ to be in the neighborhood $N_{\delta,V}(p)$ of a point $p \in \mathbf{R}^D$ even if it is arbitrarily far away from p.

LEMMA 5.1. *Let V be a set of points in $\mathbf{R}^D$, and let $p, q \in V$. Consider a degraded δ-grid $\mathcal{DG}_{\delta,V}$.*

(N.1') *If $q \notin N_{\delta,V}(p)$, then $d(p, q) > \delta$.*

(N.2') *If $q \in N_{\delta,V}(p)$, then $d(p, q) \leq 4D\delta$.*

(N.3') $q \in N_{\delta,V}(p) \iff p \in N_{\delta,V}(q)$.

LEMMA 5.2. *Let $0 < \delta' \leq \delta''/4$ be real numbers. Consider a degraded δ'-grid $\mathcal{DG}_{\delta',V'}$ and a degraded δ''-grid $\mathcal{DG}_{\delta'',V''}$, and let $p, q \in V'$. Then*

(N.4') $q \in N_{\delta',V'}(p) \implies q \in N_{\delta'',V''}(p)$.

(N.5') *For any signature $\Psi \in \{-1, 0, 1\}^D$, let $q \in b^{\Psi}_{\delta',V'}(p)$. Then $q \in N^{\Psi}_{\delta'',V''}(p)$.*

Proof. Refer to the proof of (N.4) and (N.5). By the organization of the degraded grid boxes in slabs, the argument carries over directly, except that we have to care about the width of the slabs. Since $p, q \in V'$, the slabs containing p and q w.r.t. each coordinate have width at most $2\delta'$. Since $\delta' \leq \delta''/4$, equations (2) and (3) hold, which proves (N.4'). Once (N.4') is proved, stating that the neighborhood in the smaller grid is contained in the neighborhood of the larger grid, (N.5') follows completely analogous to (N.5) by equations (4) and (5), because these equations hold by the definition of the hyperplanes employed in the proof. See Figure 3. □

Now we are ready to define our degraded grid-based sparse partition. Let $g_i := \delta_i/16D$. We store the set S_i in a degraded g_i-grid $\mathcal{DG}_{g_i,S_i}$. Analogously to equation (1) for standard grids, we define

$$S_i' := \{p \in S_i : p \text{ sparse in } \mathcal{DG}_{g_i,S_i} \text{ relative to } S_i\}. \tag{7}$$

The sparse set S_i' will also be stored in a degraded g_i-grid $\mathcal{DG}_{g_i,S_i}$. Defining the sets S_i' for each i by equation (7) yields a definition of a sparse partition analogous to the one given in Definition 3.1 for the grid case.

We adapt the abstract definition of the sparse partition (Definition 2.1) to degraded grids by changing "$\delta_i/2$" to "$\delta_i/4$" and "$\delta_i/4D$" to "$\delta_i/16D$." The bounds in Lemma 2.1 then become $\delta_i \leq \delta_{i+1}/4$ and $\delta_L/16D \leq \delta(S) \leq \delta_L$, respectively. The constants in the other lemmas of section 2 are changed analogously. Using a proof completely analogous to the one of Lemma 3.2, using (N.1')–(N.3') instead of (N.1)–(N.3), we get Lemma 5.3.

LEMMA 5.3. *Using the definition for S_i' given in equation* (7)*, we get a sparse partition according to Definition* 2.1*, with the constants changed as outlined above.*

The degraded grid-based data structure. For each $1 \leq i \leq L$, we have

- the pivot $p_i \in S_i$, its nearest neighbor q_i in S_i, and $\delta_i = d(p_i, q_i)$,
- S_i stored in a degraded g_i-grid $\mathcal{DG}_{g_i,S_i}$,
- S_i' stored in a degraded g_i-grid $\mathcal{DG}_{g_i,S_i}$,
- the heap H_i.

Now let us examine the update algorithms. In section 4, we defined the sets $down_i$. The definition remains the same here, with the notion of neighborhood in degraded grids. The *down* sets describe the point movements between the levels of the sparse partition during an insertion. Similarly, the *up* sets contain the points that move to a different level during a deletion.

Due to the point movements between the levels, the defining set of the degraded grid at level i may contain extra points additional to the ones of S_i. We therefore use a distinguished name for the defining set of the degraded grid at level i; we call it V_i.

When the insertion algorithm reaches level i without having yet performed a rebuilding, it brings along the points of $down_{i-1}$ and the new point q, see section 4. Therefore, $V_i = S_i \cup down_{i-1} \cup \{q\}$. It is important to see that, as for the sets S_i, we have

$$V_1 \supseteq V_2 \supseteq \cdots \supseteq V_L. \tag{8}$$

In the deletion algorithm, no additional point is introduced at any level except in the sparse sets S_i'. We therefore have $V_i = S_i$. (Points which vanish from level i because they move upward may be deleted from the defining set V_i at the end of the deletion algorithm.)

Remark. The defining set V_i may differ from the nonsparse set at level i only *during an update algorithm.* After completion of an update operation, these sets are

equal. In particular, the update algorithms maintain the degraded grid $\mathcal{DG}_{\delta,S_i}$ for *both* S_i *and* S_i'. That is, a point p which is new in S_i due to an update is also added to the degraded δ-grid storing the sparse set S_i', even if it is not contained in S_i'.

In the remainder of this section, we discuss the analysis of the insertion algorithm. The crucial point is the estimate on the size of the *down* sets. We transfer the relevant results to the degraded grid case, using the adapted neighborhood properties (N.4'), (N.5') given in Lemma 5.2. These properties hold with a restriction to the defining set of the degraded grid, whereas the original properties (N.4), (N.5) were valid without restriction to any point set. The nesting property (8) allows us to carry over the results nevertheless.

Analogously to the grid case, we use the following convention to describe neighborhoods in the sparse partition. For any point p, we let $N_i(p) := N_{g_i,V_i}(p)$. We use the analogous notation for the neighborhood relative to a set.

For the following statements, let $(S_i, S_i', p_i, q_i, \delta_i), 1 \le i \le L$, be a sparse partition as defined above, where V_i denotes the defining set of the degraded grid at level i.

The results corresponding to Corollary 3.1 and Lemma 3.4, obtained using (N.4'), are as follows.

- For any $1 \le i < j \le L$ and any $p \in V_j$, $\quad N_j(p, V_j) \subseteq N_i(p, V_i)$.
- For any $p \in (S \setminus S_{i+1}) \cap V_i, 1 \le i < L$, $\quad N_i(p, S \cap V_i) = \emptyset$.

Now assume that algorithm Insert(q) processes the levels $1, \ldots, i$ without a rebuilding, and the sets $down_j, 1 \le j \le i$, are defined according to equation (6). The corresponding statement to Lemma 4.3, which is obtained by using the above two statements, follows.

(9) Let $p \in down_j$ for a $j \in \{1, \ldots, i\}$. Then $p \in N_j(q)$ and $N_j(p, S \cap V_j) = \emptyset$.

With these preparations, we can prove that Lemma 4.4 remains valid; i.e.,

$$\Big| \bigcup_{1 \le j \le i} down_j \Big| \le 3^D.$$

We recall the proof of Lemma 4.4 together with the changes that are needed. The proof is now done with (9) and (N.5') replacing Lemma 4.3 and (N.5), respectively.

Assume that $p \in down_j$ for some $j \le i$. Then $p \in N_j(q)$ and $N_j(p, S \cap V_j) = \emptyset$ by (9). Let $\Psi \in \{-1, 0, 1\}^D$ be a signature such that $p \in b_j^\Psi(q)$. Refer to Figures 2 and 3. Note that the boxes $b_j^\Psi(q)$ and $b_j(q)$ have side lengths between g_j and $2g_j$ in the degraded g_j-grid, because both p and q are in V_j. The partial neighborhood $N_j^\Psi(q)$ is equal to $N_j(q) \cap N_j(p)$. Since $N_j(p, S \cap V_j) = \emptyset$ by (9), $N_j^\Psi(q)$ contains no point of $(S \cap V_j) \setminus \{p\}$.

Now consider a point $p' \in down_\ell$ for any $\ell > j$. Then $p', q \in V_\ell$, and we have $p' \in N_\ell(q)$ by (9). Assume that $p' \in b_\ell^\Psi(q)$. Since $\delta_\ell \le \delta_{j+1} \le \delta_j/4$, (N.5') gives $p' \in N_j^\Psi(q)$. We also have $p' \in V_j$, because $p' \in V_\ell$ and $V_\ell \subseteq V_j$ by the nesting property (8). However, we know from above that $N_j^\Psi(q)$ contains no point of $S \cap V_j$ except p. Therefore, $p' = p$.

This shows that, for each signature $\Psi \in \{-1, 0, 1\}^D$, all boxes $b_j^\Psi(q), 1 \le j \le i$, together contribute at most one element to the union $\bigcup_{1 \le j \le i} down_j$, which completes the proof.

Now let us turn to the running time of the algorithm. In Theorem 4.1, the box dictionary, which stores the indices of the nonempty grid boxes, was implemented using perfect hashing. Clearly, we can also store these indices in a balanced binary

search tree. Since identifying the box containing a given point now takes $O(\log n)$ time anyway on a structure of size n, the cost of searching for that box in the box dictionary (also $O(\log n)$) may be ignored. As we are now doing in logarithmic time what previously took constant time, the overall running time is also increased by a logarithmic factor. We conclude with Theorem 5.1.

THEOREM 5.1. *There exists a data structure which stores a set S of n points in $\mathbf{R}^D$ such that the minimal distance $\delta(S)$ can be found in $O(1)$ time, and all point pairs attaining $\delta(S)$ can be reported in time proportional to their number. The expected size of the structure is $O(n)$, and we can maintain the data structure as S is modified by insertions or deletions of arbitrary points in $O(\log^2 n)$ expected time per update. The data structure is randomized and fits in the algebraic decision tree model. The time bounds are obtained assuming that the updates do not depend upon the random choices made by the data structure.*

Note that the degraded grid not only depends on g_i, as in the grid case, but also on the set S_i stored in it (respectively, on the set $V_i \supset S_i$ during the insertion algorithm). Actually, it even depends on the way S_i has developed by updates. This means that this data structure no longer has the property that its distribution is independent of the history of updates. This does not affect the analysis of the algorithms, however.

6. Extensions. The data structure, as described so far, uses $O(n)$ *expected* space. In this section, we give a variant of the data structure that achieves linear space in the worst case. The update time bounds on this structure are amortized in that they will bound the expected running time of a sequence of update operations. We also show that this variant executes an update sequence quickly with high probability.

6.1. A data structure with linear space in the worst case. Recall that the space requirements are bounded by the sum of the sizes of the nonsparse sets $S_1, \ldots, S_L$ of the sparse partition, whose expected value was shown to be $O(n)$. To turn this into a worst-case bound, our first step is to slightly modify the algorithm *Sparse_Partition* given in section 2.

The modification is as follows: after picking the pivot of the grid randomly we determine the set of sparse points induced by this random choice. If at least half of the points are sparse, we call the pivot *good* and retain it as the pivot. Otherwise, we discard it and make a new random choice, continuing this process until a good pivot is found. We then continue on to the next set, making sure it has a good pivot as well, etc. Note that if all of the pivots in the sparse partition are good then, for all i, $|S_i'| \geq |S_i|/2$ so $|S_{i+1}| < |S_i|/2$, $\sum_{j>i} |S_j| = O(|S_i|)$ and the data structure will use $O(n)$ space. Furthermore, since $|S_{i+1}| \leq |S_i|/2$, the sparse partition has only $O(\log n)$ levels.

Note that at least half of the elements of a set are good pivots, and so at most two trials are needed on average until a good pivot is found. Using the same data structures as before to implement the sparse partition, we can easily prove that Lemma 6.1 holds.

LEMMA 6.1. *Let S be a set of n points in $\mathbf{R}^D$. The modified version of algorithm Sparse_Partition produces a sparse partition for S of worst-case size $O(n)$ in $O(n)$ expected time.*

Note that the above lemma only discusses creating the sparse partition itself and does not discuss creating the auxiliary data structures. However, this additional work can be completed within the same time bound, and therefore we do not discuss it. In the next section, in which we discuss high probability bounds, we will need to distinguish between constructing the sparse partition itself on the one hand and the

complete data structure on the other. For the remainder of this section we assume that the procedures *Build* and *Near_Build* have been modified as above to produce sparse partitions of worst-case linear size.

We now move on to the update algorithms. The idea will be to maintain a sparse partition all of whose pivots are good or at least not too far from being good. We make the following observations.

1. The uniformity property (Definition 2.2) is lost. However, since at least half of the elements are good pivots, the probability of an element being the pivot right after the construction of a new sparse partition is $2/n$ for a set of size n.
2. Updates can gradually unbalance the data structure in the sense that more than a constant proportion of the elements at that level can become non-sparse. In the earlier version of the algorithm the uniformity condition guaranteed that the data structure was, probabilistically, well balanced. Lacking the uniformity condition we enforce the balance of the sparse partition "by hand" to ensure that the data structure uses linear space. That is, we count the number of update operations that affect a level of the data structure, rebuilding after this count has reached a suitable constant fraction of the cardinality of the set at that level at the time of the last rebuilding. This will ensure that $|S_i'|$ is always at least some constant proportion of $|S_i|$. In the sequel we call these rebuildings *amortized* to distinguish them from the *probabilistic* rebuildings that can be caused by a particular update.

We now sketch the modifications to the update algorithms, using the notation from section 4. We associate two variables, $last_i$ and $count_i$, with level i of the current sparse partition. $last_i$ equals $|S_i|$ after the last (amortized or probabilistic) rebuilding that affected S_i, i.e., the last rebuilding performed at a level $j \leq i$. $count_i$ denotes the number of update operations since the last rebuilding that affected level i. In what follows, $c > 1$ is a sufficiently large constant.

In the insertion algorithm (Figure 7), let q be the newly inserted point, and suppose we are at level i, $down_{i-1}$ has been computed, and $\widetilde{S}_i = S_i \cup down_{i-1} \cup \{q\}$. Only step 2 (checking for rebuild) is changed as follows:

1. $count_i := count_i + 1$; **if** $count_i \geq last_i/c$ **then** $Build(\widetilde{S}_i, i)$; stop;
2. **if** q or an element of $down_{i-1}$ is closer to the pivot p_i than its previous nearest neighbor
   then $Build(\widetilde{S}_i, i)$; stop;

The first item is the amortized rebuilding discussed above, and the second item is one component of the probabilistic rebuilding in the original algorithm. Note that the first part of that rebuild step—which ensured the uniformity of the pivots by probabilistically deciding whether to make one of the elements in $down_{i-1} \cup \{q\}$ the new pivot—does not appear here. Since we do not have complete uniformity here anyway (bad elements cannot be pivots), we treat a newly inserted element as if it were a bad element.

In the deletion algorithm (Figure 10), suppose again that we are at level i, up_{i-1} has been computed, and we have $\widetilde{S}_i = (S_i \setminus up_{i-1}) \setminus \{q\}$. Only step 2 is changed as follows:

1. $count_i := count_i + 1$; **if** $count_i \geq last_i/c$ **then** $Build(\widetilde{S}_i, i)$; stop;
2. **if** q or an element of up_{i-1} is either the pivot p_i or its nearest neighbor q_i
   then $Build(\widetilde{S}_i, i)$; stop;

It is clear that the above modifications do not affect the correctness of the update

algorithms. We now analyze its cost. First note that immediately after a rebuilding $|S_i'| \geq |S_i|/2$. A single update may only add or subtract a constant number of items to or from S_i and S_i'. Thus if c is taken to be large enough, the amortized rebuildings guarantee that $|S_i'| > |S_i|/4$ always holds, ensuring that (i) the data structure has $O(\log n)$ levels and (ii) the size of the structure is $O(n)$. (Note that the constant number of points that can be moved at each step depends upon the dimension D, so c must be chosen dependent upon D as well.)

As long as no rebuilding occurs, the algorithm uses a constant number of dictionary operations at each level. Since there are $O(\log n)$ levels the total expected cost of all dictionary operations is $O(\log n)$. The heap updates cost $O(\log n)$ time as before. Thus the total expected time required for an update operation which does not perform a rebuilding is $O(\log n)$.

Recall now that there are two types of rebuildings: amortized and probabilistic. We start by analyzing the amortized rebuildings. Note that an amortized rebuilding occurs on a set S_i of size m only if a sequence of $\Theta(m)$ updates has occurred without a rebuilding of S_i. Since a rebuilding costs $O(m)$ expected time each of the sequence of $\Theta(m)$ updates incurs an $O(1)$ amortized cost for the rebuilding of S_i. Summing over all $O(\log n)$ levels in the data structure gives an amortized expected cost of $O(\log n)$ per update operation.

For the probabilistic rebuildings, we will show that for each update which affects a set S_i of size m, the probability of a rebuilding is at most c'/m for some fixed constant c'. Since the expected cost of a rebuilding is $O(m)$ this means that the expected cost of rebuilding a set at a given level at any particular step will be $O(1)$. Summing over all $O(\log n)$ levels yields a total expected cost of $O(\log n)$ per update.

To show that the probability of a rebuilding is at most c'/m for some c' we note that immediately after a rebuilding the probability of rebuilding a set S_i of size m is c'/m where $c' = 2$. Between rebuildings, an adversary gains knowledge about the possible pivots of the data structure permitting him to force a rebuilding with an appropriately chosen insert or delete command. However, we have already seen that an adversary can only exclude at most a constant number of points from being a pivot in each update operation. Thus, if c is taken to be large enough that after m/c updates at a level of initial size m, the adversary still must consider $|S_i|/4$ possible good pivots, this implies that the probability of a probabilistic rebuilding (i.e., item 2 of the modified updates) is still $O(1/m)$. This proves the following theorem.

THEOREM 6.1. *The data structure of Theorem* 4.1 *can be modified to use $O(n)$ worst-case space. The modified data structure has $O(\log n)$ amortized expected update time, and the time complexities of all other operations are unchanged.*

Applying the same modifications to the data structure of Theorem 5.1 we get Theorem 6.2.

THEOREM 6.2. *The data structure of Theorem* 5.1 *can be modified to use $O(n)$ worst-case space. The modified data structure has $O(\log^2 n)$ amortized expected update time, and the time complexities of all other operations are unchanged.*

6.2. High probability bounds. The previous theorem yields the expected running time of the dynamic closest-pair algorithm on a sequence of updates. We now discuss the probability that the running time of the data structure of Theorem 6.2 on such an update sequence deviates significantly from its expectation. In what follows we say that an event occurs with *n-polynomial probability* if, for any fixed $s > 0$, it occurs with probability $1 - O(n^{-s})$. A process is said to take $O(f(n))$ time with n-polynomial probability if, for any fixed $s > 0$, the process takes at most $c(s)f(n)$ time

for sufficiently large n, where $c(s)$ is a constant dependent upon s, with probability $1 - O(n^{-s})$.

THEOREM 6.3. *Let S be a set of n points in $\mathbf{R}^D$. The data structure of Theorem 6.2 performs a sequence of n updates on S in $O(n \log^2 n)$ time with n-polynomial probability.*

Remark. We can also prove that the data structure of Theorem 6.1 processes a sequence of $\Theta(n)$ updates, starting with a set of size n, in $O(n \log^2 n)$ time with n-polynomial probability; see [18] for details.

Proof. First note that the theorem is obviously correct if we only count the costs of the part of the updates that do not include rebuildings. The nonrebuilding part of the update consists of $O(1)$ heap operations, each costing $O(\log n)$ time for a total of $O(\log n)$ time, and of $O(\log n)$ dictionary operations each costing $O(\log n)$ time for a total of $O(\log^2 n)$ time. (Recall that our data structure has $O(\log n)$ levels in the worst case for a set of size n.) Hence, if no rebuilding occurs, an update takes $O(\log^2 n)$ time in the worst case.

We now analyze the rebuilding cost and show that rebuilding a sparse partition for a set of size m will require $O(m \log n)$ time with n-polynomial probability. First note that during the rebuilding of a level the chance of a random pivot being a good one is at least $1/2$; the probability of not finding a good pivot after k trials is therefore at most 2^{-k}, so with n-polynomial probability only $O(\log n)$ pivots need to be checked before finding a good one. At first sight, it appears as if even checking if a single pivot is good should take $O(m \log n)$ time, since checking to see if a point is sparse appears to require $O(1)$ queries to the box dictionary, each costing $O(\log n)$ time.

However, if we only want to check if a pivot is good, we can avoid queries to the box dictionary as follows: during the building of the degraded grid data structure, we can link each nonempty box with the nonempty boxes in its neighborhood, where we mean the occurrences of the boxes in the box dictionary (not only in the geometric representation, which is trivial). We can use these pointers to find the sparse points of the point set which is being stored *in linear time*, as follows: walk through the list of nonempty boxes of the grid. With the help of the above-described pointers, we can access the point lists associated with the neighboring boxes in constant time, replacing the queries in the box dictionary. (This faster method was not mentioned earlier as computing the sparse points of a set was never previously a bottleneck.) Thus we can identify the sparse points and decide whether the pivot is good in $O(m)$ time; it is only in the computing of the restricted distances where we will need $\Theta(m \log n)$ time.

We conclude that a good pivot can be found and the grid for that level built and processed in $O(m \log n)$ time with n-polynomial probability. Since the levels decrease geometrically in size, good pivots for all succeeding levels of the partition can also be found in $O(m \log n)$ time with n-polynomial probability. The remainder of the *Build* procedure takes $O(m \log n)$ time as before.

We now analyze the amortized and probabilistic rebuildings separately, studying the amortized ones first. Actually, using the same reasoning as developed in the previous subsection, we find that an amortized rebuilding of set S_i of size m is only performed if $\Theta(m)$ previous updates did not rebuild S_i. Now rebuilding S_i requires $O(m \log n)$ time with n-polynomial probability, so with n-polynomial probability the amortized rebuilding cost per update of S_i is $O(\log n)$. Since there are $O(\log n)$ levels, this adds up to $O(n \log^2 n)$ time with n-polynomial probability.

Before analyzing the cost of the probabilistic rebuildings, i.e., the rebuildings caused by a deletion of a pivot or insertion/deletion of the nearest neighbor to a pivot,

it will help to quickly review what we are trying to study. As we start with a data structure which contains n points and then make n updates to it, the data structure contains at most $2n$ items at any update step. Since $|S_i'| > |S_i|/4$ we have $|S_{i+1}| < 3|S_i|/4$ so the data structure never contains more than $L = \log_{4/3}(2n) = \Theta(\log n)$ levels. Now suppose that $|S_i| = m$. Then as described in the previous subsection S_i is rebuilt with probability $O(1/m)$. The cost of rebuilding S_i will be $O(m \log n)$ with n-polynomial probability. Let $m_{i,t}$ denote the size of S_i at the end of update $t-1$ for $1 \le i \le L$ and $1 \le t \le n$. If S_i does not exist at the end of update $t-1$ set $m_{i,t} = 1$ by convention. Now define the random variables

$$X_{i,t} = \begin{cases} m_{i,t} & \text{with probability } 1/m_{i,t}, \\ 0 & \text{with probability } 1 - 1/m_{i,t}. \end{cases}$$

Then the cost of probabilistically rebuilding S_i at update t, counting both degraded grid and heap operations, is $O(X_{i,t} \log n)$ with n-polynomial probability by the above discussion. Hence the total cost of probabilistic rebuildings over all updates is bounded by $O(M \log n)$ with n-polynomial probability, where $M = \sum_{i,t} X_{i,t}$. We now show that M itself is $O(n \log n)$ with n-polynomial probability. The proof of Theorem 6.3 will follow.

The first complication we encounter in bounding M is that the $X_{i,t}$ are *not* independent because the $m_{i,t}$ depend upon each other. Nevertheless, since $|S| \le 2n$ and the amortized rebuilding ensures that $|S_{i+1}| < 3|S_i|/4$, the relation

$$m_{i,t} \le \max\{1, 2 \cdot (3/4)^{i-1} \cdot n\} \tag{10}$$

must always hold. We sidestep the dependence issue by showing that, for *any* values of $m_{i,t}$ that satisfy equation (10), M has n-polynomial probability of being $O(n \log n)$. This allows us to assume that we are given some *fixed* (but arbitrary) values $m_{i,t}$ which satisfy (10), and that the variables $X_{i,t}$ are as defined above and *independent* of each other.

The following lemma is obtained by a straightforward modification of the proof of the Chernoff bound given in [14, p. 68].

LEMMA 6.2. *Let $Y_1, \ldots, Y_k$ be independent random variables, and let $a_1, \ldots, a_k \in [1, A]$ for some $A \ge 1$. For $i = 1, \ldots, k$ suppose that*

$$Y_i = \begin{cases} a_i & \textit{with probability } 1/a_i, \\ 0 & \textit{with probability } 1 - 1/a_i, \end{cases}$$

and let $Y = \sum_{i=1}^{k} Y_i$. Then for any $\delta > 0$,

$$\Pr[Y > (1+\delta)k] < \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^{k/A}. \tag{11}$$

Proof. Let $\lambda = (\ln(1+\delta))/A$ and note that $\lambda > 0$. Then

$$\begin{aligned} \Pr[Y > (1+\delta)k] &= \Pr[e^{\lambda Y} > e^{\lambda(1+\delta)k}] \\ &\le \frac{\mathrm{E}(e^{\lambda Y})}{e^{\lambda(1+\delta)k}} \\ &= \frac{\prod_{i=1}^{k} \mathrm{E}(e^{\lambda Y_i})}{e^{\lambda(1+\delta)k}} \\ &= \frac{\prod_{i=1}^{k} (\frac{1}{a_i} e^{\lambda a_i} + 1 - \frac{1}{a_i})}{e^{\lambda(1+\delta)k}}. \end{aligned}$$

However, $f(x) = e^{\lambda x}/x + 1 - 1/x$ can easily be seen to be an increasing function of x, for $x > 0$. Therefore,

$$\Pr[Y > (1+\delta)k] \le \frac{\left(\frac{1}{A}e^{\lambda A} + 1 - \frac{1}{A}\right)^k}{e^{\lambda(1+\delta)k}} \le \frac{(1+\delta/A)^k}{(1+\delta)^{(1+\delta)k/A}} < \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^{k/A}. \qquad \square$$

Let s be a given positive integer. We apply Lemma 6.2 to $X_i = \sum_{t=1}^{n} X_{i,t}$ for each $1 \le i \le 5 \log\log n$, noting that $m_{i,t} \le 2n$ for this range of i. By choosing $\delta = c \cdot \log n / \log\log n - 1$ for sufficiently large $c > 0$ and $A = 2n$, we obtain that, for all $1 \le i \le 5 \log\log n$,

$$\Pr[X_i > c \cdot n \cdot \log n / \log\log n] < n^{-s-1}. \tag{12}$$

We now apply Lemma 6.2 to $X_i = \sum_{t=1}^{n} X_{i,t}$ for each $5 \log\log n < i \le L$, noting that $m_{i,t} \le n/\log n$ for this range of i. By choosing $\delta = c' - 1$ for sufficiently large $c' > 1$ and $A = n/\log n$, we obtain that, for all $5 \log\log n < i \le L$,

$$\Pr[X_i > c' \cdot n] < n^{-s-1}. \tag{13}$$

From (12) and (13) we conclude that $M = \sum_{i=1}^{n} X_i = O(n \log n)$ with probability at least $1 - n^{-s}$, and hence $M = O(n \log n)$ with n-polynomial probability. $\square$

7. Concluding remarks. In this paper, we have given the first solution to the fully dynamic closest pair problem which achieves linear space and polylogarithmic update time simultaneously. After a preliminary version of this paper was published, Kapoor and Smid [13] achieved this goal with a deterministic method which has amortized update time $O(\log^{D-1} n \log\log n)$ for $D \ge 3$ and $O(\log^2 n/(\log\log n)^{\ell})$ for the case $D = 2$, where ℓ is an arbitrary nonnegative integer constant. Also, Callahan and Kosaraju [4] gave a deterministic data structure achieving $O(\log^2 n)$ update time while using linear space for any fixed dimension. Finally, Bespamyatnikh [3] gave an optimal deterministic solution for the dynamic closest pair problem: for any fixed dimension, his technique achieves $O(\log n)$ update time and uses $O(n)$ space.

REFERENCES

[1] M. Ben-Or, *Lower bounds for algebraic computation trees*, in Proc. 15th Ann. ACM Sympos. Theory Comput., 1983, pp. 80–86.

[2] J. L. Bentley and M. I. Shamos, *Divide-and-conquer in multidimensional space*, in Proc. 8th Ann. ACM Sympos. Theory Comput., 1976, pp. 220–230.

[3] S. N. Bespamyatnikh, *An optimal algorithm for closest pair maintenance*, in Proc. 11th Ann. ACM Sympos. Comput. Geom., 1995, pp. 152–161.

[4] P. B. Callahan and S. R. Kosaraju, *Algorithms for dynamic closest-pair and n-body potential fields*, in Proc. 6th ACM-SIAM Sympos. Discrete Algorithms, 1995, pp. 263–272.

[5] J. L. Carter and M. N. Wegman, *Universal classes of hash functions*, J. Comput. System Sci., 18 (1979), pp. 143–154.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. McGraw–Hill, New York, 1990.

[7] M. T. Dickerson, R. L. Drysdale, and J. R. Sack, *Simple algorithms for enumerating interpoint distances and finding k nearest neighbors*, Internat. J. Comput. Geom. Appl., 2 (1992), pp. 221–239.

[8] W. H. E. Day and H. Edelsbrunner, *Efficient algorithms for agglomerative hierarchical clustering methods*, J. Classification, 1 (1984), pp. 7–24.

[9] A. Datta, H.-P. Lenhof, C. Schwarz, and M. Smid, *Static and dynamic algorithms for k-point clustering problems*, J. Algorithms, 19 (1995), pp. 474–503.

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

[11] M. J. Golin, R. Raman, C. Schwarz, and M. Smid, *Simple randomized algorithms for closest pair problems*, Nordic J. Comput., 2 (1995), pp. 3–27.
[12] S. Khuller and Y. Matias, *A simple randomized sieve algorithm for the closest-pair problem*, Inform. and Comput., 118 (1995), pp. 34–37.
[13] S. Kapoor and M. Smid, *New techniques for exact and approximate dynamic closest-point problems*, SIAM J. Comput., 25 (1996), pp. 775–796.
[14] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, London, 1995.
[15] W. Pugh, *Skip lists: A probabilistic alternative to balanced trees*, Comm. Assoc. Comput. Mach., 33 (1990), pp. 668–676.
[16] M. O. Rabin, *Probabilistic algorithms*, in Algorithms and Complexity, J. F. Traub, Ed., Academic Press, New York, 1976, pp. 21–30.
[17] J. S. Salowe, *Enumerating interdistances in space*, Internat. J. Comput. Geom. Appl., 2 (1992), pp. 49–59.
[18] C. Schwarz, *Data Structures and Algorithms for the Dynamic Closest Pair Problem*, Ph.D. thesis, Universität des Saarlandes, Saarbrücken, Germany, 1993.
[19] R. Seidel and C. R. Aragon, *Randomized search trees*, Algorithmica, 16 (1996), pp. 464–497.
[20] M. I. Shamos and D. Hoey, *Closest-point problems*, in Proc. 16th Ann. IEEE Sympos. Found. Comput. Sci., 1975, pp. 151–162.
[21] M. Smid, *Maintaining the minimal distance of a point set in less than linear time*, Algorithms Rev., 2 (1991), pp. 33–44.
[22] M. Smid, *Maintaining the minimal distance of a point set in polylogarithmic time*, Discrete Comput. Geom., 7 (1992), pp. 415–431.
[23] C. Schwarz, M. Smid, and J. Snoeyink, *An optimal algorithm for the on-line closest-pair problem*, Algorithmica, 12 (1994), pp. 18–29.
[24] K. J. Supowit, *New techniques for some dynamic closest-point and farthest-point problems*, in Proc. 1st ACM-SIAM Sympos. Discrete Algorithms, 1990, pp. 84–90.

SIAM J. COMPUT.
Vol. 27, No. 4, pp. 1073–1082, August 1998
© 1998 Society for Industrial and Applied Mathematics
008

SEPARATING EXPONENTIALLY AMBIGUOUS FINITE AUTOMATA FROM POLYNOMIALLY AMBIGUOUS FINITE AUTOMATA*

HING LEUNG†

Abstract. We resolve an open problem raised by Ravikumar and Ibarra [*SIAM J. Comput.*, 18 (1989), pp. 1263–1282] on the succinctness of representations relating to the types of ambiguity of finite automata. We show that there exists a family of nondeterministic finite automata $\{A_n\}$ over a two-letter alphabet such that, for any positive integer n, A_n is exponentially ambiguous and has n states, whereas the smallest equivalent deterministic finite automaton has 2^n states, and any smallest equivalent polynomially ambiguous finite automaton has $2^n - 1$ states.

Key words. nondeterministic finite automata, ambiguity, succinctness of representation

AMS subject classification. 68Q68

PII. S0097539793252092

1. Introduction.

In their 1989 paper [RI89] Ravikumar and Ibarra raised some interesting questions relating the type of ambiguity of finite automata to the succinctness in their number of states. They considered the following five classes of finite automata: deterministic finite automata (DFA), nondeterministic finite automata (NFA), unambiguous NFA (UFA), finitely ambiguous NFA (FNA), and polynomially ambiguous NFA (PNA). Formal definitions of these classes are given in section 2 of this paper.

Let C_1 and C_2 be any two of the above five classes of finite automata. We say that C_1 can be polynomially converted to C_2 (written $C_1 \leq_P C_2$) if there exists a polynomial p such that for any finite automaton in C_1 with n states we can find an equivalent finite automaton in C_2 with at most $p(n)$ states. C_1 is said to be polynomially related to C_2 (written $C_1 =_P C_2$) if $C_1 \leq_P C_2$ and $C_2 \leq_P C_1$. C_1 is said to be separated from C_2 if $C_1 \neq_P C_2$. We write $C_1 <_P C_2$ if $C_1 \leq_P C_2$ and $C_1 \neq_P C_2$.

It is immediate that DFA $\leq_P$ UFA, UFA $\leq_P$ FNA, FNA $\leq_P$ PNA, and PNA $\leq_P$ NFA.

The following is known: DFA $<_P$ NFA [MF71], [Mo71], DFA $<_P$ UFA [Sc78], [SH85], [RI89], UFA $<_P$ FNA [Sc78], [RI89], and UFA $<_P$ NFA [SH85]. It is unknown whether FNA $<_P$ NFA. Ravikumar and Ibarra conjecture that FNA $<_P$ PNA and that PNA $<_P$ NFA [RI89].

In this paper, we prove that PNA $<_P$ NFA which immediately implies that FNA $<_P$ NFA. The other conjecture that FNA $<_P$ PNA still remains open.

In summary, we have

$$\text{DFA} <_P \text{UFA} <_P \text{FNA} \leq_P \text{PNA} <_P \text{NFA}.$$

Specifically, we show that there exists a family of NFAs $\{A_n \mid n \geq 1\}$ over a two-letter alphabet such that, for any positive integer n, A_n is exponentially ambiguous

* Received by the editors July 14, 1993; accepted for publication (in revised form) May 31, 1996; published electronically May 19, 1998. This research was supported by an Alexander von Humboldt research fellowship and was done while the author was visiting the University of Frankfurt, Germany. A preliminary version of the paper appeared in the proceedings of ISAAC'93.

http://www.siam.org/journals/sicomp/27-4/25209.html

† Department of Computer Science, New Mexico State University, Las Cruces, NM 88003 (hleung@cs.nmsu.edu).

and has n states, whereas the smallest equivalent DFA has 2^n states and any smallest equivalent PNA has $2^n - 1$ states.

Our results show that any PNA equivalent to A_n cannot do better in the number of states than the smallest equivalent DFA obtained by the subset construction except for the saving of the dead state.

Another way to interpret our results is as follows: let us first define that an NFA is "strongly" ambiguous if there is a useful state q (that is, q can be reached from some starting state and can reach some final state) and a string w such that M can process w starting from state q and ending also with state q in more than one way. Then by a characterization in [IR86] our results show that A_n is strongly ambiguous with n states, whereas the smallest equivalent DFA has 2^n states and any smallest equivalent NFA that is not strongly ambiguous has $2^n - 1$ states.

Section 2 presents the definitions and some basic results. Section 3 presents the family of NFAs $\{A_n\}$ and proves the main result of this paper by a series of lemmas.

2. Preliminaries. We assume that the reader is familiar with the basic definitions and notations in finite automata theory and the Myhill–Nerode theorem [HU79] as well as the basics of graph theory [Ha69].

Let w be a string, and L a language. We denote by w^R the reverse of w and by L^R the set of strings v^R where $v \in L$.

Throughout this paper, we assume a model of NFA that is slightly more general than the one defined in [HU79] in that we allow a set of starting states instead of only one starting state. Thus, an NFA M is a 5-tuple $(Q, \Sigma, \delta, Q_I, Q_F)$ where Q is the set of states, Σ is the alphabet set, $\delta : Q \times \Sigma \longrightarrow 2^Q$ is the transition function, Q_I is the set of starting states, and Q_F is the set of final states.

Given an NFA M, we define the ambiguity of a string w to be the number of different accepting paths for w in M. Note that a string w is in the language of M if and only if the ambiguity of w is not zero. The ambiguity function $amb_M : \mathbb{N}_0 \longrightarrow \mathbb{N}_0$ is defined such that $amb_M(n)$ is the maximum of the ambiguities of strings that are of length n or less. Remark: amb_M is nondecreasing.

M is called unambiguous if the ambiguity of any string is either zero or one. M is called finitely (respectively, polynomially, exponentially) ambiguous if amb_M can be bounded by a constant (respectively, polynomial, exponential) function f; that is, for all $n \in \mathbb{N}_0$, $amb_M(n) \leq f(n)$.

It is easy to see that $amb_M(n) \leq s_1 s^n$ where s_1 is the cardinality of Q_I and s is the cardinality of Q. Thus, every NFA must be exponentially ambiguous.

M is called strictly exponentially ambiguous [IR86] if M is exponentially ambiguous but not polynomially ambiguous. It is known [IR86] that M is strictly exponentially ambiguous if and only if there is a useful state q and there is a string w such that M can process w starting from state q and ending also with state q in more than one way.

3. Main result. For any positive integer n, we define an NFA $A_n = (Q, \Sigma, \delta, \{q_1\}, \{q_1\})$ where $Q = \{q_1, q_2, \ldots, q_n\}$, q_1 is the only starting state and the only final state, $\Sigma = \{0, 1\}$, and δ (see Figure 1) is defined as follows:

- $\delta(q_1, 0) = \{q_1, q_2\}$,
- $\delta(q_i, 0) = \{q_{i+1}\}$ for $2 \leq i \leq n-1$,
- $\delta(q_n, 0) = \{q_1\}$,
- $\delta(q_1, 1) = \emptyset$,
- $\delta(q_i, 1) = \{q_i\}$ for $2 \leq i \leq n$.

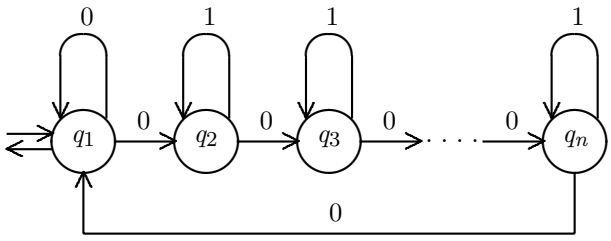


FIG. 1. *Transition diagram of* A_n.

We denote the language of A_n by L_n, which is $(0 + (01^*)^{n-1}0)^*$. It is easy to see that $L_n = L_n^R = L_n^*$.

It is argued in section 2 that every NFA is exponentially ambiguous. Thus A_n is exponentially ambiguous. Moreover, A_n is strictly exponentially ambiguous since the ambiguity of 0^m is at least $2^{\lfloor m/n \rfloor}$.

We will prove that any DFA recognizing L_n has at least 2^n states (Lemma 2), any UFA recognizing L_n has at least $2^n - 1$ states (Lemma 4), and any PNA recognizing L_n also has at least $2^n - 1$ states (Theorem 1).

First, we present some definitions. Given a language L and a string x, $\mathit{prefix}(L) \stackrel{\text{def}}{=} \{w \mid \exists w',\ ww' \in L\}$ and $x^{-1}(L) \stackrel{\text{def}}{=} \{w \mid xw \in L\}$. The two operations *prefix* and x^{-1} commute since both $x^{-1}(\mathit{prefix}(L))$ and $\mathit{prefix}(x^{-1}(L))$ equal $\{w \mid \exists w',\ xww' \in L\}$.

Let *kill_non_*q_1 denote $0(10)^{n-1}$, *accept* denote 0^{n-1}, and *reset* denote *accept kill_non_*q_1. The intuitive concepts of the strings *kill_non_*q_1, *accept*, and *reset* are reflected in the following properties: for any $P \subseteq Q$, $\delta(P, \mathit{kill\_non\_}q_1) = P - \{q_2, \ldots, q_n\}$. For any nonempty subset $P \subseteq Q$, observe that $q_1 \in \delta(P, \mathit{accept})$ and $\delta(P, \mathit{reset}) = \{q_1\}$. Equivalently, for any $x \in \mathit{prefix}(L_n)$, $x\ \mathit{accept} \in L_n$ and $(x\ \mathit{reset})^{-1}(L_n) = L_n$.

For any $P \subseteq Q$, let $w_P \in \Sigma^*$ be $w_1 0 w_n 0 w_{n-1} 0 \ldots 0 w_1$ where $w_i = \epsilon$ if $q_i \in P$ and $w_i = 1$ otherwise; and let $u_P \in \Sigma^*$ be $0^{n-1}\ w_P$. The meaning of w_P and u_P can be understood as the strings that satisfy Lemma 1 and Corollary 1, respectively, given below. The following properties (Lemma 1 and Corollaries 1 and 2) of the strings w_P and u_P are very crucial in proving the main theorem of the paper.

LEMMA 1. *For any* $P \subseteq Q$, *we have*

(1) $\delta(P, w_P) = P$;

(2) *for any* $q \in P$, $\delta(q, w_P) \supseteq \{q\}$;

(3) $\delta(Q - P, w_P) = \emptyset$.

Proof. The proof of Lemma 1, which is quite long but straightforward, is given in the appendix. ∎

COROLLARY 1. *For any* $P \subseteq Q$, $\delta(q_1, u_P) = P$.

Proof. $\delta(q_1, u_P) = \delta(q_1, 0^{n-1}\ w_P) = \delta(Q, w_P) = \delta(P, w_P) \bigcup \delta(Q - P, w_P) = \delta(P, w_P) = P$ by parts (1) and (3) of Lemma 1. ∎

COROLLARY 2. *For any* $P, P' \subseteq Q$, $u_P w_{P'} \in \mathit{prefix}(L_n)$ *if and only if* $P \bigcap P' \neq \emptyset$.

Proof. Suppose $P \bigcap P' \neq \emptyset$. Let $q \in P \bigcap P'$. Then $\delta(q_1, u_P w_{P'}) = \delta(P, w_{P'}) \supseteq \delta(q, w_{P'}) \supseteq \{q\} \neq \emptyset$ by Corollary 1 and part (2) of Lemma 1. Since all states in A_n are useful, $u_P w_{P'} \in \mathit{prefix}(L_n)$.

Suppose $P \bigcap P' = \emptyset$. Then $P \subseteq Q - P'$ and $\delta(q_1, u_P w_{P'}) = \delta(P, w_{P'}) \subseteq \delta(Q - P', w_{P'}) = \emptyset$ by Corollary 1 and part (3) of Lemma 1. Thus we have that $u_P w_{P'} \notin \mathit{prefix}(L_n)$. ∎

With the basic properties established, we begin to prove the first result that the smallest DFA recognizing L_n has 2^n states.

LEMMA 2. *The smallest DFA recognizing L_n has 2^n states.*

Proof. By Corollary 1, all subsets of states can be realized in the subset construction. We show that any two different subsets of states are not equivalent; then we are done by the Myhill–Nerode theorem. Let $P \neq P' \subseteq Q$. Let q_i be the state with the largest subscript such that q_i belongs only to exactly one of P and P'. That is, for any q_j where $i+1 \leq j \leq n$, either q_j belongs to both P and P' or q_j does not belong to any one of P and P'. If $i = 1$, then P and P' can be distinguished by the empty string. Assume that $1 < i \leq n$. Then P and P' can be distinguished by $(10)^{n+1-i}$. □

The next result (Lemma 4) that we want to establish is that a smallest UFA recognizing L_n cannot do better in the number of states than a smallest DFA besides the saving of the dead state. Some technical definitions and a technical lemma (Lemma 3) are needed first.

Let M_n be a $2^n - 1 \times 2^n - 1$ matrix over the field of characteristic 2 with rows and columns indexed by the nonempty subsets of Q such that $M_n(P, P') = 1$ if $u_P\ w_{P'}\ accept \in L_n$, and $M_n(P, P') = 0$ otherwise. By Corollary 2 and the property of *accept*, $M_n(P, P') = 1$ if $P \bigcap P' \neq \emptyset$ and $M_n(P, P') = 0$ otherwise.

LEMMA 3. *The rank of M_n is $2^n - 1$.*

Proof. Equivalently, we can index rows and columns of M_n by n-bit positive binary numbers in the order of increasing values such that any n-bit positive binary number $b_n b_{n-1} \ldots b_1$ corresponds to the nonempty subset $P \subseteq Q$ with the property that for any $1 \leq i \leq n$, $q_i \in P$ if and only if $b_i = 1$. Note that the indices range from binary number of value 1 to binary number of value $2^n - 1$. Thus, $M_n(\alpha, \beta) = 1$ if there is some i such that the ith bits of α and β are both 1, and $M_n(\alpha, \beta) = 0$ otherwise.

We are going to show by induction on n that M_n has rank $2^n - 1$. For $n = 1$, then $M_n = [1]$, which is a 1×1 matrix of rank $2^1 - 1 = 1$.

Suppose that the statement is true for $n = k$. Consider $n = k + 1$. We observe that the matrix M_{k+1} (see Figure 2) can be characterized as follows:

- The matrix M_{k+1} is symmetric. (Reason: According to the definition given above for $M_n(\alpha, \beta)$, it is immediate that $M_n(\alpha, \beta) = M_n(\beta, \alpha)$.)
- The middle row and the middle column both have $2^k - 1$ zeros followed by 2^k ones. (Reason: The middle row has an index of a one followed by k zeros. On the other hand, the first $2^k - 1$ column indices are binary numbers that begin with zero, and the last 2^k column indices are binary numbers that begin with one. Thus, by definition, the middle row has $2^k - 1$ zeros followed by 2^k ones. Since M_{k+1} is symmetric, the middle column also has $2^k - 1$ zeros followed by 2^k ones.)
- Separated by the middle row and the middle column, we have four square submatrices of sizes $2^k - 1 \times 2^k - 1$ each. Let us name the upper-left, upper-right, lower-left, and lower-right submatrices as UL, UR, LL, and LR. Then UL, UR, and LL are the same as M_k, and LR is a matrix of ones. (Reason: Elements in LR correspond to row and column indices that both begin with one. Thus LR is a matrix of ones. For elements in other submatrices, either the row or column index begins with zero. Thus the element values are determined by the remaining k bits of the row and column indices, which behave in the same way as the row and column indices of M_k. Hence, UL,

$$M_{k+1} = \left[\begin{array}{c|c|c} UL & \begin{matrix}0\\ \vdots\\ 0\end{matrix} & UR \\ \hline 0\cdots 0 & 1 & 1\cdots 1 \\ \hline LL & \begin{matrix}1\\ \vdots\\ 1\end{matrix} & LR \end{array}\right] = \left[\begin{array}{c|c|c} M_k & \begin{matrix}0\\ \vdots\\ 0\end{matrix} & M_k \\ \hline 0\cdots 0 & 1 & 1\cdots 1 \\ \hline M_k & \begin{matrix}1\\ \vdots\\ 1\end{matrix} & \begin{matrix}1\cdots 1\\ \vdots\\ 1\cdots 1\end{matrix} \end{array}\right]$$

FIG. 2. *Structure of M_{k+1}.*

$$M_{k+1} = \left[\begin{array}{c|c|c} UL & \begin{matrix}0\\ \vdots\\ 0\end{matrix} & UR \\ \hline 0\cdots 0 & 1 & 1\cdots 1 \\ \hline LL & \begin{matrix}0\\ \vdots\\ 0\end{matrix} & LR \end{array}\right] = \left[\begin{array}{c|c|c} M_k & \begin{matrix}0\\ \vdots\\ 0\end{matrix} & \begin{matrix}0\cdots 0\\ \vdots\\ 0\cdots 0\end{matrix} \\ \hline 0\cdots 0 & 1 & 1\cdots 1 \\ \hline M_k & \begin{matrix}0\\ \vdots\\ 0\end{matrix} & M_k \end{array}\right]$$

FIG. 3. *Structure of M_{k+1} (after transformations).*

UR, and LL are the same as M_k.)

We want to apply elementary row operations to show that M_{k+1} can be transformed to an identity matrix. By subtracting the middle row from each of the rows in the lower half of the matrix, LL remains unchanged whereas the rest of the entries in the lower half become all zeros. Note that the middle column now has all zeros except a one in the middle. Let us swap the lower half of the matrix with the upper half. See Figure 3 for the current structure of M_{k+1}.

By induction, we can apply elementary row operations to the upper half such that UL becomes an identity matrix, and the rest of the entries in the upper half are zeros. With UL being the identity matrix and zeros elsewhere in the upper half, we apply again elementary row operations so that LL becomes all zeros whereas LR remains the same as M_k. See Figure 4 for the current structure of M_{k+1}.

Next, again by induction, we can transform LR to identity matrix since the rest of the entries in the lower half are all zeros. Finally subtract each of the rows in the lower half from the middle row; the middle row becomes a row with all zeros except a one in the middle position. Therefore, an identity matrix is obtained and the rank of M_{k+1} is thus $2^{k+1}-1$. □

LEMMA 4. *A smallest UFA recognizing L_n has 2^n-1 states.*

Proof. First, by removing the dead state from the DFA obtained by the subset construction, we have a UFA with 2^n-1 states. Next we are going to use a technique introduced in [Sc78] to show that any UFA would require at least 2^n-1 states.

Let U be a UFA recognizing L_n with the finite set of states denoted by K. Let R be a matrix over the field of characteristic 2 with rows indexed by K and columns indexed by the nonempty subsets of Q such that $R(k,P)=1$ if U can reach a final state starting from state $k \in K$ on consuming w_P *accept*, and $R(k,P)=0$ otherwise.

We claim that any row in M_n is a linear combination of the rows in R. Given a nonempty subset P of Q, let $K' \subseteq K$ be the set of states reached by U from the set of starting states on consuming u_P. For any $k_1 \neq k_2 \in K'$ and for any nonempty subset

$$M_{k+1} = \left[\begin{array}{c|c|c} UL & \begin{matrix}0\\ \vdots\\ 0\end{matrix} & UR \\ \hline 0\cdots 0 & 1 & 1\cdots 1 \\ \hline LL & \begin{matrix}0\\ \vdots\\ 0\end{matrix} & LR \end{array}\right] = \left[\begin{array}{c|c|c} I_k & \begin{matrix}0\\ \vdots\\ 0\end{matrix} & \begin{matrix}0\cdots 0\\ \vdots\\ 0\cdots 0\end{matrix} \\ \hline 0\cdots 0 & 1 & 1\cdots 1 \\ \hline \begin{matrix}0\cdots 0\\ \vdots\\ 0\cdots 0\end{matrix} & \begin{matrix}0\\ \vdots\\ 0\end{matrix} & M_k \end{array}\right]$$

FIG. 4. *Structure of M_{k+1} (after transformations).*

P' of Q, $R(k_1, P')$ and $R(k_2, P')$ cannot both have value one; that is, at most one of $R(k, P')$, for $k \in K'$, is 1. Otherwise, there are two different accepting paths for $u_P w_{P'}$ *accept* in U, which contradicts the assumption that U is unambiguous. Thus, the row indexed by P in M_n is the sum of the rows indexed by K' in R.

Therefore, the rank of M_n is less than or equal to the rank of R. Hence, K must have at least $2^n - 1$ states so that the rank of R is at least $2^n - 1$. □

We need Lemma 5 below to prove Lemma 6, which is the main technical lemma that helps us to show that a smallest PNA recognizing L_n has $2^n - 1$ states (Theorem 1). In fact, Lemma 5 can be viewed as a generalization of Lemma 4.

LEMMA 5. *Any UFA recognizing L such that prefix(L) = prefix(L_n) requires at least $2^n - 1$ states.*

Proof. Since L is regular, there exists a finite set of strings $\{\gamma_1, \ldots, \gamma_h\} \subseteq \Sigma^*$ such that for any $z \in$ *prefix*(L), exactly one of $z\gamma_i$, for $1 \le i \le h$, is in L.

Let X be a $2^n - 1 \times h(2^n - 1)$ matrix over the field of characteristic 2 with rows indexed by the nonempty subsets of Q and columns indexed by $\{(P, i) \mid \emptyset \ne P \subseteq Q,\ 1 \le i \le h\}$ such that $X(P, (P', i)) = 1$ if $u_P w_{P'} \gamma_i \in L$, and $X(P, (P', i)) = 0$ otherwise.

We claim that the rank of X is $2^n - 1$. We define X' to be a $2^n - 1 \times 2^n - 1$ matrix over the field of characteristic 2 with rows and columns indexed by the nonempty subsets of Q such that the column indexed by P is the sum of the h columns in X indexed by $\{(P, i) \mid 1 \le i \le h\}$.

We want to show that X' is the same matrix as M_n. That is, we want to show that $X'(P, P') = 1$ if $P \bigcap P' \ne \emptyset$ and $X'(P, P') = 0$ otherwise.

Suppose $P \bigcap P' \ne \emptyset$. By Corollary 2, $u_P w_{P'} \in$ *prefix*(L_n) = *prefix*(L). By the definition of γ_i's, exactly one of $u_P w_{P'} \gamma_i$, for $1 \le i \le h$, is in L. That is, exactly one of $X(P, (P', i))$, for $1 \le i \le h$, is 1. Hence, $X'(P, P') = 1$.

Suppose $P \bigcap P' = \emptyset$. By Corollary 2, $u_P w_{P'} \notin$ *prefix*(L_n) = *prefix*(L). Therefore, $u_P w_{P'} \gamma_i \notin L$ for $1 \le i \le h$. Hence, $X(P, (P', i)) = 0$ for $1 \le i \le h$ and $X'(P, P') = 0$.

By Lemma 3, the rank of X' is $2^n - 1$. Since each column of X' is obtained by a linear combination of the columns of X, the rank of X must be bigger than or equal to the rank of X'. Thus, the rank of X is at least $2^n - 1$. Moreover since the number of rows in X is $2^n - 1$, the rank of X is at most $2^n - 1$. Therefore, the rank of X is $2^n - 1$.

Finally, since the rank of X is $2^n - 1$ and by using the same technique as in the proof of Lemma 4, a smallest UFA for L must have at least $2^n - 1$ states. □

The following lemma is a generalization of Lemma 5, which is the special case where $x = \epsilon$.

LEMMA 6. *Let U be a UFA with the number of states less than 2^n-1 which accepts L such that prefix$(L) \subseteq$ prefix(L_n). Then for all $x \in$ prefix(L_n), $x^{-1}($prefix$(L)) \subset x^{-1}($prefix$(L_n))$.*

Proof. From prefix$(L) \subseteq$ prefix(L_n), it is immediate that $x^{-1}($prefix$(L)) \subseteq x^{-1}($prefix$(L_n))$ for any arbitrary x. We want to show that $x^{-1}($prefix$(L)) \neq x^{-1}($prefix$(L_n))$ for any $x \in$ prefix(L_n).

Suppose to the contrary that $x^{-1}($prefix$(L)) = x^{-1}($prefix$(L_n))$ for some $x \in$ prefix(L_n). Thus, $z^{-1}($prefix$(L)) = z^{-1}($prefix$(L_n))$ where $z = x$ *reset*. Since z^{-1} and *prefix* commute, prefix$(z^{-1}(L)) =$ prefix$(z^{-1}(L_n))$. Let L' be $z^{-1}(L)$. Then we obtain prefix$(L') =$ prefix(L_n) since $z^{-1}(L_n) = L_n$ by the property of *reset*.

We are going to construct a UFA U' with less than $2^n - 1$ states to recognize L' which is a contradiction, because of Lemma 5.

The transition diagram for U' is the same as that of U. The set of starting states for U' is defined to be the set of states reached by U on consuming z from the set of starting states of U. The set of final states for U' is again the same as that of U. It is clear that the language accepted by U' is $L' = z^{-1}(L)$. Also, U' cannot be ambiguous otherwise U is also ambiguous. Moreover, the number of states in U' is the same as the number of states in U; therefore, it is less than $2^n - 1$. ∎

We are ready to prove the main result of this paper.

THEOREM 1. *A smallest PNA recognizing L_n has $2^n - 1$ states.*

Proof. By removing the dead state from the DFA obtained by the subset construction, we have a UFA, which is polynomially ambiguous, with $2^n - 1$ states.

Let M be a PNA for L_n with the smallest number of states. Then every state in M must be useful.

Consider the transition diagram of M. Since the strongly connected components form a partial ordering with respect to reachability, there must exist one strongly connected component, denoted T, that cannot be reached from other strongly connected components.

We claim that T must have at least $2^n - 1$ states. Suppose to the contrary that it has less than $2^n - 1$ states.

T must have some starting states in it. Otherwise it is not useful which contradicts the definition of M. Let the set of starting states of M that appears in T be $\{p_1, \ldots, p_k\}$.

Let $1 \leq i \leq k$. We define an NFA T_{p_i} such that p_i is now the only starting and final state, and the transition diagram for T_{p_i} is T. We want to check that Lemma 6 can be applied to the language of T_{p_i}. First, T_{p_i} has less than $2^n - 1$ states. Next, T_{p_i} is a UFA; otherwise by the characterization given in section 2, M is strictly exponentially ambiguous, a contradiction. Let $w \in$ prefix$(L(T_{p_i}))$. Then p_i must reach a nonempty subset of states in T on consuming w. Since all states in M are useful, w is therefore in prefix(L_n). Hence, prefix$(L(T_{p_i})) \subseteq$ prefix(L_n).

Consider the set of UFAs $\{T_{p_i} \mid 1 \leq i \leq k\}$. Let $x_0 = \epsilon \in$ prefix(L_n). For $1 \leq i \leq k$, we define x_i to be a string chosen arbitrarily from

$$(x_0 x_1 \ldots x_{i-1})^{-1}(\text{prefix}(L_n)) - (x_0 x_1 \ldots x_{i-1})^{-1}(\text{prefix}(L(T_{p_i})).$$

Thus, $x_0 x_1 \ldots x_i \in$ prefix(L_n) for $0 \leq i \leq k$. The existence of x_i, for $1 \leq i \leq k$, is then guaranteed by Lemma 6. Let x be $x_1 \ldots x_k$. By the way x_i's are defined, $x \notin$ prefix$(L(T_{p_i}))$ for $1 \leq i \leq k$. Thus, each UFA T_{p_i}, $1 \leq i \leq k$, reaches the empty set from the starting state p_i on consuming x since all states in T_{p_i} are useful.

Let $z = x$ *reset*. Since $x \in$ *prefix*(L_n), then $z^{-1}(L_n) = L_n$ by the property of *reset*.

Let us consider M again. From the set of starting states, M reaches a subset of states, denoted P, on consuming z. Since $z \in$ *prefix*(L_n) and $L(M) = L_n$, P is not empty. Moreover, by the previous discussions, P does not include any state in T.

We define another NFA M' by removing the set of states in T from the state set of M and let P be the new set of starting states, whereas the set of final states is the set of final states of M minus the set of states in T. By the facts that M accepts L_n and $z^{-1}(L_n) = L_n$, M' must also accept L_n. But this is a contradiction since M' is now a PNA accepting L_n with a smaller number of states than M.

Therefore, T cannot have less than $2^n - 1$ states. Hence, M has at least $2^n - 1$ states. □

Remark. We can prove the same results for the related family of automata B_n, defined as the same as A_n except that all states in B_n are final.

Appendix (Proof of Lemma 1).

LEMMA 1. *For any $P \subseteq Q$, we have*

(1) $\delta(P, w_P) = P$;

(2) *for any $q \in P$, $\delta(q, w_P) \supseteq \{q\}$*;

(3) $\delta(Q - P, w_P) = \emptyset$.

Proof. Given a state $q_i \in Q$, we define *shift*(q_i) to be q_{i+1} if $1 \leq i \leq n-1$, and q_1 if $i = n$. Given $P \subseteq Q$, we extend the definition of *shift* such that *shift*$(P) = \{$*shift*$(q) \mid q \in P\}$. Moreover, for any $q \in Q$ and $P \subseteq Q$, we define *shift*$^0(q) = q$ and *shift*$^0(P) = P$. Note that *shift*$^n(q) = q$ and *shift*$^n(P) = P$.

We begin by proving part (1) of the lemma that $\delta(P, w_P) = P$.

Observe that $\delta(P, w_1) = P$. This is because if $q_1 \in P$, then $w_1 = \epsilon$ and $\delta(P, w_1) = \delta(P, \epsilon) = P$. Otherwise if $q_1 \notin P$, then $w_1 = 1$ and $\delta(P, w_1) = \delta(P, 1) = P$ since $q_1 \notin P$.

Therefore, $\delta(P, w_P) = \delta(P, 0w_n0w_{n-1} \ldots 0w_1)$.

We want to show by induction that $\delta(P, 0w_n0w_{n-1} \ldots 0w_{n-i+1}) =$ *shift*$^i(P)$ for $0 \leq i \leq n$. Then we are done since $\delta(P, 0w_n0w_{n-1} \ldots 0w_1) =$ *shift*$^n(P) = P$.

Base. $i = 0$. $\delta(P, \epsilon) = P =$ *shift*$^0(P)$.

Induction hypothesis. Assume that the statement is true for $0 \leq k \leq n-1$.

Induction step. By induction, we have $\delta(P, 0w_n0w_{n-1} \ldots 0w_{n-k+1}0w_{n-k}) = \delta(\delta(P, 0w_n0w_{n-1} \ldots 0w_{n-k+1}), 0w_{n-k}) = \delta($*shift*$^k(P), 0w_{n-k})$. The induction proof is completed if we can verify that $\delta($*shift*$^k(P), 0w_{n-k}) =$ *shift*$^{k+1}(P)$.

Case 1. ($q_{n-k} \in P$). Then $w_{n-k} = \epsilon$ and $q_n \in$ *shift*$^k(P)$. Thus, $\delta($*shift*$^k(P), 0w_{n-k})$ $= \delta($*shift*$^k(P), 0) =$ *shift*$^{k+1}(P)$ since $q_n \in$ *shift*$^k(P)$.

Case 2. ($q_{n-k} \notin P$). Then $w_{n-k} = 1$ and $q_n \notin$ *shift*$^k(P)$. Thus, $\delta($*shift*$^k(P), 0w_{n-k})$ $= \delta($*shift*$^k(P), 01) =$ *shift*$^{k+1}(P)$ since $q_n \notin$ *shift*$^k(P)$.

We finish the proof of part (1) of the lemma. Next, we prove part (2) of the lemma that for any $q \in P$, $\delta(q, w_P) \supseteq \{q\}$.

Observe that $\delta(q, w_1) = \{q\}$. This is because if $q_1 \in P$, then $w_1 = \epsilon$ and $\delta(q, w_1) = \delta(q, \epsilon) = \{q\}$. Otherwise if $q_1 \notin P$, then $w_1 = 1$ and $\delta(q, w_1) = \delta(q, 1) = \{q\}$ since $q \neq q_1$ by the facts that $q_1 \notin P$ and $q \in P$.

Therefore, $\delta(q, w_P) = \delta(q, 0w_n0w_{n-1} \ldots 0w_1)$.

We want to show by induction that $\delta(q, 0w_n0w_{n-1} \ldots 0w_{n-i+1}) \supseteq$ *shift*$^i(\{q\})$ for $0 \leq i \leq n$. Then we are done since $\delta(q, 0w_n0w_{n-1} \ldots 0w_1) \supseteq$ *shift*$^n(\{q\}) = \{q\}$.

Base. $i = 0$. $\delta(q, \epsilon) = \{q\} =$ *shift*$^0(\{q\})$. Thus, $\delta(q, \epsilon) \supseteq$ *shift*$^0(\{q\})$.

Induction hypothesis. Assume that the statement is true for $0 \leq k \leq n-1$.

Induction step. By induction, we have $\delta(q, 0w_n0w_{n-1}\dots0w_{n-k+1}0w_{n-k}) = \delta(\delta(q, 0w_n0w_{n-1}\dots0w_{n-k+1}), 0w_{n-k}) \supseteq \delta(shift^k(\{q\}), 0w_{n-k})$. The induction proof is complete if we can verify that $\delta(shift^k(\{q\}), 0w_{n-k}) \supseteq shift^{k+1}(\{q\})$.

Case 1. ($q_n = shift^k(q)$). Then $w_{n-k} = \epsilon$ since $q_{n-k} = q \in P$. Thus, $\delta(shift^k(\{q\}), 0w_{n-k}) = \delta(shift^k(\{q\}), 0) = \delta(q_n, 0) = \{q_1\} = shift(\{q_n\}) = shift(shift^k(\{q\})) = shift^{k+1}(\{q\})$. Hence, $\delta(shift^k(\{q\}), 0w_{n-k}) \supseteq shift^{k+1}(\{q\})$.

Case 2. ($q_n \neq shift^k(q)$). Then no matter whether $w_{n-k} = \epsilon$ or $w_{n-k} = 1$, we always have $\delta(shift^k(\{q\}), 0w_{n-k}) \supseteq shift^{k+1}(\{q\})$.

We finish the proof of part (2) of the lemma. Finally, we prove part (3) of the lemma that $\delta(Q - P, w_P) = \emptyset$.

We claim that $\delta(Q-P, w_1 0w_n0w_{n-1}0\dots0w_2) = \emptyset$. Hence, $\delta(Q-P, w_P) = \delta(Q-P, w_1 0w_n0w_{n-1}0\dots0w_20w_1) = \delta(\delta(Q-P, w_1 0w_n0w_{n-1}0\dots0w_2), 0w_1) = \delta(\emptyset, 0w_1) = \emptyset$.

First observe that $\delta(Q - P, w_1) = Q - P - \{q_1\}$. This is because if $q_1 \in P$ then $w_1 = \epsilon$ and $q_1 \notin Q-P$. Thus $\delta(Q-P, w_1) = \delta(Q-P, \epsilon) = Q-P = Q-P-\{q_1\}$ since $q_1 \notin Q - P$. Otherwise if $q_1 \notin P$ then $w_1 = 1$. Thus $\delta(Q - P, w_1) = \delta(Q - P, 1) = Q - P - \{q_1\}$.

Next we want to show by induction that for $1 \leq i \leq n$, $\delta(Q-P-\{q_1\}, 0w_n0w_{n-1}\dots 0w_{n-i+2}) = shift^{i-1}(Q-P)-\{q_1, q_2, \dots, q_i\}$. Then we are done since by taking $i = n$, we have $\delta(Q - P - \{q_1\}, 0w_n0w_{n-1}\dots0w_2) = shift^{n-1}(Q - P) - \{q_1, q_2, \dots, q_n\} = \emptyset$.

Base. $i = 1$. $\delta(Q - P - \{q_1\}, \epsilon) = Q - P - \{q_1\} = shift^0(Q - P) - \{q_1\}$.

Induction hypothesis. Assume that the statement is true for $1 \leq k \leq n - 1$.

Induction step. By induction, we have $\delta(Q-P-\{q_1\}, 0w_n0w_{n-1}\dots0w_{n-k+2}0w_{n-k+1}) = \delta(shift^{k-1}(Q - P) - \{q_1, q_2, \dots, q_k\}, 0w_{n-k+1})$. The induction proof is completed if we can verify that $\delta(shift^{k-1}(Q - P) - \{q_1, q_2, \dots, q_k\}, 0w_{n-k+1}) = shift^k(Q - P) - \{q_1, q_2, \dots, q_k, q_{k+1}\}$.

Case 1. ($q_{n-k+1} \in P$). Then $w_{n-k+1} = \epsilon$ and $q_n \notin shift^{k-1}(Q - P)$. Thus, $shift^{k-1}(Q-P) = shift^{k-1}(Q-P)-\{q_n\}$. Hence, $\delta(shift^{k-1}(Q-P)-\{q_1, q_2, \dots, q_k\}, 0w_{n-k+1}) = \delta(shift^{k-1}(Q - P) - \{q_n, q_1, q_2, \dots, q_k\}, 0) = shift^k(Q - P) - \{q_1, q_2, \dots, q_k, q_{k+1}\}$.

Case 2. ($q_{n-k+1} \notin P$). Then $w_{n-k+1} = 1$. Thus, $\delta(shift^{k-1}(Q-P)-\{q_1, q_2, \dots, q_k\}, 0w_{n-k+1}) = \delta(shift^{k-1}(Q-P)-\{q_1, q_2, \dots, q_k\}, 01) = shift^k(Q-P)-\{q_1, q_2, \dots, q_k, q_{k+1}\}$.

We finish the proof of the claim and hence part (3) of the lemma. □

Acknowledgments. The author thanks Andreas Weber for many fruitful discussions and comments throughout this work. He also thanks Jonathan Goldstine and Detlef Wotschke for their valuable discussions.

REFERENCES

[Ha69] F. Harary, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.

[HU79] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, MA, 1979.

[IR86] O. Ibarra and B. Ravikumar, *On sparseness, ambiguity and other decision problems for acceptors and transducers*, in Proc. 3rd Annual Symposium on Theoretical Aspects of Computer Science, Orsay, France, Lecture Notes in Computer Science 210, 1986, pp. 171–179.

[MF71] A. Meyer and M. Fischer, *Economy of description by automata, grammars, and formal systems*, in Proc. 12th Symposium on Switching and Automata Theory, 1971, pp. 188–191.

[Mo71] F. Moore, *On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata*, IEEE Trans. Comput., 20 (1971), pp. 1211–1214.

[RI89] B. Ravikumar and O. Ibarra, *Relating the type of ambiguity of finite automata to the succinctness of their representation*, SIAM J. Comput., 18 (1989), pp. 1263–1282.

[Sc78] E. Schmidt, *Succinctness of Descriptions of Context-Free, Regular, and Finite Languages*, Ph.D. Thesis, Cornell University, Ithaca, NY, 1978.

[SH85] R. Stearns and H. Hunt, *On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata*, SIAM J. Comput., 14 (1985), pp. 598–611.

SIAM J. COMPUT.
Vol. 27, No. 4, pp. 1083–1098, August 1998
© 1998 Society for Industrial and Applied Mathematics
009

AN $\Omega(\sqrt{\log \log n})$ LOWER BOUND FOR ROUTING IN OPTICAL NETWORKS *

LESLIE ANN GOLDBERG†, MARK JERRUM‡, AND PHILIP D. MACKENZIE§

Abstract. Optical communication is likely to significantly speed up parallel computation because the vast bandwidth of the optical medium can be divided to produce communication networks of very high degree. However, the problem of contention in high-degree networks makes the routing problem in these networks theoretically (and practically) difficult. In this paper we examine Valiant's *h-relation routing problem*, which is a fundamental problem in the theory of parallel computing. The h-relation routing problem arises both in the direct implementation of specific parallel algorithms on distributed-memory machines and in the general simulation of shared-memory models such as the PRAM on distributed-memory machines. In an h-relation routing problem each processor has up to h messages that it wishes to send to other processors and each processor is the destination of at most h messages. We present a lower bound for routing an h-relation (for any $h > 1$) on a complete optical network of size n. Our lower bound applies to any randomized distributed algorithm for this task. Specifically, we show that the expected number of communication steps required to route an arbitrary h-relation is $\Omega(h + \sqrt{\log \log n})$. This is the first known lower bound for this problem which does not restrict the class of algorithms under consideration.

Key words. parallel algorithms, randomized algorithms, routing, optical networks

AMS subject classifications. 68Q22, 68R05

PII. S0097539794272569

1. Introduction.

In current distributed-memory parallel computers, a number of processors equipped with private local memory communicate by sending messages via a network of communication links. Current technology restricts the network to be of low degree: each processor in the network can communicate directly with only a few others, and the remainder must be reached indirectly by routing messages along a sequence of links. The emerging technology of optical communication challenges the assumption that the network must be of low degree. In particular, the huge bandwidth of the optical medium can be divided so that each processor has its own channel for receiving messages and each processor can send on any channel. Even though such an interconnection network is a complete graph, there remains the problem of contention: no processor can receive messages simultaneously from two other processors without corruption. The problem of avoiding contention is much more difficult in high-degree networks (such as optical networks) than in traditional low-degree networks.

The problem of routing in optical networks is captured mathematically by the OCPC model. In an n-processor *completely connected Optical Communication Paral-*

*Received by the editors August 8, 1994; accepted for publication June 7, 1996; published electronically May 19, 1998. A preliminary version of this paper appeared in the *Proceedings of the 6th Annual ACM Symposium on Parallel Algorithms and Architectures*, Cape May, NJ, 1994.

http://www.siam.org/journals/sicomp/27-4/27256.html

†Department of Computer Science, University of Warwick, Coventry CV4 7AL UK (leslie@dcs.warwick.ac.uk). This work was performed while the author was at Sandia National Laboratories and was supported by the U.S. Department of Energy under contract DE-AC04-76DP00789.

‡Department of Computer Science, The University of Edinburgh, The King's Buildings, Edinburgh EH9 3JZ UK (mrj@dcs.ed.ac.uk). This work was supported in part by grant GR/F 90363 of the UK Science and Engineering Research Council, and Esprit Working Group "RAND" and was partly done while the author was visiting the NEC Research Institute, Princeton, NJ.

§Department of Mathematics and Computer Science, Boise State University, Boise, ID 83725 (philmac@cs.idbsu.edu). This work was performed while the author was at the University of Texas and was supported by Texas Advanced Research Projects Grant 003658480.

lel Computer (n-OCPC) n processors with local memory are connected by a complete network. A computation on this computer consists of a sequence of communication steps. During each communication step each processor can perform some local computation and then send one message to any other processor. If a processor is sent a single message during a communication step then it receives this message successfully, but if it is sent more than one message then the transmissions are garbled and it receives none of them.

Eshaghian [5, 6] first studied the computational aspects of parallel architectures with complete optical interconnection networks. The OCPC model is an abstract model of computation which formalizes important properties of such architectures. It was first introduced by Anderson and Miller [1] and Eshaghian and Kumar [7] and has subsequently been studied by several authors including Valiant [22], Geréb-Graus and Tsantilas [12], and Gerbessiotis and Valiant [11] (though not always under the name OCPC). Aside from its importance as a model for optical communication, the OCPC has the attraction of being a clean, mathematically appealing model that allows us to study a single issue, namely the resolution of contention between independent processors, in isolation from other factors. It has recently been observed that the n-processor OCPC is equivalent to an ERCW PRAM with n global memory cells. Thus our results carry over to that model. For details, see [20].

In this paper we study a fundamental communication problem for multiprocessor computers: that of routing h-relations. This problem arises both in the direct implementation of specific parallel algorithms [1], and in the simulation of shared-memory models, such as the PRAM, on more realistic distributed-memory models [22]. An *h-relation* routing problem [22] is a communication problem in which each processor has up to h messages that it wishes to send to other processors. The destinations of these messages can be arbitrary except that each processor is the destination of at most h messages. The goal is to design a fast algorithm for the n-OCPC that can route an arbitrary h-relation.

Anderson and Miller [1] have observed that an h-relation can easily be routed in h communication steps if all of the processors are given *total* information about the h-relation to be routed. A more interesting (and more realistic) situation arises if we assume that each processor initially knows only about the messages that it wants to send, and that processors learn about the rest of the h-relation only through receiving messages from other processors. This is the usual assumption, and the one that will be made here.

Valiant [22], building on work of Anderson and Miller [1], developed a randomized algorithm that routes an arbitrary h-relation in $O(h + \log n)$ steps, on average. Subsequently, Goldberg, Jerrum, Leighton, and Rao [13] presented a more complex randomized algorithm for the same task that runs in $O(h + \log \log n)$ steps and has failure probability $n^{-\alpha}$ for any constant α. The latter algorithm is asymptotically the fastest known, and it would be interesting to discover whether it is the best possible. Our attention therefore turns to lower bounds.

Goldberg et al. [13] proved a lower bound for a restricted class of algorithms known as *direct*, in which a processor may only send messages directly to their final destination. (Thus the only freedom a processor has is in its choice of *when* to attempt to send its messages.) They proved that for any (randomized) direct algorithm there is a 2-relation that takes $\Omega(\log n)$ steps to route with success probability $\frac{1}{2}$, thus showing that even in a completely connected network it is advantageous to route messages indirectly. Subsequently, MacKenzie, Plaxton, and Rajaraman [19] generalized this

result by showing that for any (randomized) direct algorithm and any $h \geq 2$ there is an h-relation that takes $\Omega(h + \log h \log n)$ expected steps to route. (This bound is tight as the randomized direct algorithm of Geréb-Graus and Tsantilas [12] routes h-relations in $O(h + \log h \log n)$ expected steps.)

Obtaining a lower bound for *unrestricted* algorithms has proved a much greater challenge, owing, no doubt, to the rich variety of strategies that are available to a nondirect algorithm. (Some of the possibilities will be glimpsed in section 2.) Indeed, no lower bound beyond the trivial $\Omega(h)$ was previously known. The new result in this paper is a lower bound on the number of steps required to route 2-relations on an n-OCPC. We prove that for any randomized algorithm there is a 2-relation such that the expected number of steps required to route the relation is $\Omega(\sqrt{\log\log n})$. (See Theorem 3.1 for a precise statement of the result.) Our result implies that for any $h > 1$ the number of steps required to route an arbitrary h-relation is $\Omega(h + \sqrt{\log\log n})$. We note that our lower bound also holds for routing $c+1$-relations in the c-collision OCPC model studied by Dietzfelbinger and Meyer auf der Heide [2].

Our proof technique is based on one used in MacKenzie [18] (see also [16]). The technique is a novel extension to the *random restriction method* which Furst, Saxe, and Sipser [10] used to prove circuit lower bounds. In this extension the choice of *which* inputs to randomly restrict at each stage depends on a careful analysis of the appropriate step of the algorithm under consideration. Although the random restriction method was first used to obtain a deterministic lower bound, we obtain a randomized lower bound by proving that the induction hypothesis in the random restriction method holds with high probability (and not merely with nonzero probability). Other randomized lower bounds were obtained by this method in Hastad [15].

The gap between the current upper and lower bounds on routing h-relations deserves comment. Section 4 indicates why a lower bound of the form $\Omega(\sqrt{\log\log n})$ is the limit of the current technique. An example presented in that section points to an issue that must be faced in any attempt to improve the current lower bound. It appears that some new idea is necessary to make further progress on this front.

2. Some preliminary observations. Imagine that two processors p and q wish to deliver a single message each to a common destination processor within $O(\log\log n)$ steps. Assume that p and q do not know each other's identity. A simple strategy is for p and q each to flip a coin and attempt to transmit its packet to the destination processor if the coin comes up "heads." After $O(\log\log n)$ steps, the probability that p and q have failed to transmit their packets is at least $(\log n)^{-O(1)}$. If $n^{\Omega(1)}$ pairs of processors simultaneously employ this strategy to deliver their messages to separate destinations, the probability that they all succeed is negligible. Some more subtle approach is required.

One possibility, suggested by Rao, is the following. Suppose the processors are assigned binary sequence numbers, and that the numbers assigned to p and q are $p_1p_2\ldots p_r$ and $q_1q_2\ldots q_r$, where $r \sim \log n$. By simultaneously sending messages to processors $p_1p_2\ldots p_{r/2}0\ldots0$ and $q_1q_2\ldots q_{r/2}0\ldots0$, respectively, processors p and q may discover whether their sequence numbers differ in the first $r/2$ bits. After about $\log\log n$ experiments of this general form, and using binary search, p and q can agree on a bit position at which their sequence numbers differ; this bit can then be used to determine a priority for the processors, and hence resolve the conflict. Note that this method (with slight modification) could be used by $n^{\Omega(1)}$ pairs of processors simultaneously. Observe that p and q are not sending messages in order to get the content of the message to another processor but to learn some information about the

competing processor.

A second strategy is replication of messages. In $O(\log \log n)$ binary replication steps, p and q can each prime a set of $\Theta(\log n)$ processors with the message they are required to transmit. These two sets of processors then use the naive coin-flipping strategy to attempt to send their cloned messages to a common target set of size $\Theta(\log n)$. In just a constant number of attempts, the probability that either a p-message or a q-message fails to get through is reduced to $n^{-\Omega(1)}$ where the implicit constant is arbitrary. Finally, the messages in the target set can be funneled into the destination processor by a procedure which is an inverse of the cloning phase. Note that the failure probability is much smaller here than for the naive strategy and can be expected to remain small when many pairs of processors simultaneously attempt to send to distinct targets.

These two examples indicate the subtle strategies that are available to indirect algorithms. With these in mind, it is possible to give a little of the flavor of the lower bound argument. After t-steps, some set of processors (of size at most exponential in t) will be aware that processor p or q has a packet to send. Viewing the situation crudely, these "agents" for p and q can act in one of two modes, or possibly a mixture: (a) they can send messages to some narrow set of destinations that is only weakly dependent on the identity of the source processor, or (b) they can send to a wide destination set, or one that is strongly dependent on the identity of the source processor.

The first strategy sketched above operates purely in mode (a), while the second strategy relies on mode (b) to recruit the processors that are required in the replication phase. The key point is that the effectiveness of mode (a) is limited by the collisions that inevitably occur, while mode (b) is limited in its ability to "advance messages toward their destination." The lower bound proof to be described in section 3 analyzes the tradeoff between these modes. That both strategies described above are effective suggests that the whole range of the tradeoff must be examined and explains some of the technical complexity of the proof.

3. The lower bound argument.

3.1. Definitions and goals. Our goal is to establish the following.

THEOREM 3.1. *Let A be a randomized algorithm that routes 2-relations on an n-OCPC. Then there is a 2-relation on which the expected number of communication steps used by A is at least $\sqrt{\log \log n}/4$.*[1]

The first step in the proof of Theorem 3.1 will be to reduce to the case of deterministic A. A certain restricted class of 2-relations (to be defined presently) will be termed "relevant." We will use a weak form of a theorem of Yao (as stated in [9]) to show that Theorem 3.1 reduces to proving the following.

THEOREM 3.2. *Let A be a deterministic algorithm that allegedly routes 2-relations in $T = \sqrt{\log \log n/2}$ steps. Let the input to A be drawn u.a.r. from the set of relevant 2-relations. Then the probability that A successfully routes the input is at most $\frac{1}{2}$.*

To define the class of relevant 2-relations, we make the following definitions, which will be explained below. (For the purpose of the proof, we define "h-relation" in a restricted way. In the h-relations that we consider, a processor can receive up to h messages but can send at most one message.)

DEFINITION. *A* partial h-relation *is a function from the set $\{1, \ldots, n\}$ of processors to $\{0, 1, *\}$.*

[1] All logarithms in this paper are to the base 2.

DEFINITION. *An* h-relation *is a partial* h*-relation in which no processor is mapped to* "$*$."

Intuitively, we think of the n-OCPC as being partitioned into $n^{4/5}$ *ranges* containing $n^{1/5}$ processors each. If an h-relation maps a processor to 1 then this processor has a message to send, and if it maps a processor to 0 then this processor does not have a message to send. The destination of each message is the first processor in the range containing the sending processor. We can now make the following definitions.

DEFINITION. *A* relevant 2-relation *is an* h*-relation in which exactly two processors in each range are mapped to* "1."

DEFINITION. *A partial* h*-relation* f *is a* refinement *of a partial* h*-relation* f' *(this is denoted by* $f \leq f'$*) if* $f'(p) = 1$ *implies* $f(p) = 1$*, and* $f'(p) = 0$ *implies* $f(p) = 0$.

DEFINITION. *A* partial relevant 2-relation *is a partial* h*-relation that has a refinement which is a relevant* 2*-relation.*

DEFINITION. f_* *is the partial* h*-relation that maps every processor to* "$*$."

3.2. Generating a random 2-relation. Algorithm RANDOMSET can be used to randomly generate a relevant 2-relation one processor at a time. It is called with a partial relevant 2-relation f and a set P of processors which are mapped to "$*$" by f. The processors in P are randomly mapped to "0" or "1" in such a way that the resulting function f' is a partial relevant 2-relation and the following claim holds.

```
Function RANDOMSET(f, P)
    Let f' = f
    For each p ∈ P
        Let s = | { q | q is in the range of p and f(q) = "*" } |
        If no processors in the same range as p are mapped to "1" by f
                With probability 2/s set f'(p) = 1
                With probability 1 − 2/s set f'(p) = 0
        If one processor in the same range as p is mapped to "1" by f
                With probability 1/s set f'(p) = 1
                With probability 1 − 1/s set f'(p) = 0
        Otherwise set f'(p) = 0
    Return f'
End RANDOMSET
```

CLAIM 3.3. *An* h*-relation* f *generated solely by calls to RANDOMSET is a relevant* 2*-relation generated uniformly at random (u.a.r.) from the set of relevant* 2*-relations.*

Proof. The proof is straightforward. ∎

3.3. Defining the knowledge set and t-good partial h-relations. Now we make some definitions that deal with the running of a deterministic algorithm A on an n-OCPC when the input is an h-relation f.

DEFINITION. *The* $(0, f)$-trace *of processor* p *is the tuple* $\langle p, f(p)\rangle$*. The* (t, f)-trace *of processor* p *(for* $t > 0$*) is the tuple* $\langle p, f(p), \lambda_1, \ldots, \lambda_t\rangle$ *in which* λ_j *is the message that processor* p *receives at step* j *if such a message exists and* λ_j *is the null symbol otherwise.*

Note that we lose no generality by assuming that if p sends a message on step t then it sends its entire $(t-1)$-trace. (Since each processor is allowed to know the algorithms that the other processors run we can simulate an algorithm which sends different messages by an algorithm which sends traces using the same pattern of communications.)

DEFINITION. *Processor p is a* direct (t, f)-receiver *of processor q if either $p = q$ or when A is run with input f, p receives a message from q in the first t steps.*

DEFINITION. *Processor p is an* indirect (t, f)-receiver *of q if either p is a direct (t, f)-receiver of q, or when A is run with input f, there is some processor k and some time step $t' < t$ such that k is an indirect (t', f)-receiver of q and p receives a message from k during steps $t' + 1, \ldots, t$.*

DEFINITION. *A set S of processors is a* (t, g)-dependency set *of a processor p if g is a partial h-relation and for any relevant 2-relations f_1 and f_2 which refine g and have $f_1(q) = f_2(q)$ for every processor $q \in S$, the (t, f_1)-trace of p is the same as the (t, f_2)-trace of p.*

The intuition behind the above definition is that p is not dependent on processors outside S, since these could not affect its trace. Note that if S' and S'' are (t, g)-dependency sets of a processor p then so is $S' \cap S''$, so p has a unique (t, g)-dependency set of minimum size.

DEFINITION. *The* (t, g)-knowledge set *of a processor p is the smallest (t, g)-dependency set of p.*

Suppose that g is a partial h-relation and that f is a relevant 2-relation which refines g. Note that if $g(p) =$ "$*$" and q has a (t, g)-dependency set which excludes p then q cannot be an indirect (t, f)-receiver of p. Also note that if $g(p) \neq$ "$*$" then p is not in the (t, g)-knowledge set of any processor. We now define the following constants and functions of n.

DEFINITION. $k_i = 3^i$, $s_0 = n^{1/5}$, $w_i = s_i^{1/k_i}/21k_i^2$, $r_i = s_i^4$, *and* $s_i = w_{i-1}^{1/7}$ (*for* $i \geq 1$).

Recall that $T = \sqrt{\log \log n/2}$. We will use the following facts.

FACT 3.4. *For large enough n and $t \leq T$, $s_t \geq 2^{\log^{1/3} n}$.*

Proof. Note that for $t \geq 1$ we have

$$s_t = (7/3)^{-1/7} 9^{-t/7} {s_{t-1}}^{1/(7 \cdot 3^{t-1})}.$$

Let $\alpha(t)$ denote $1/(7^t 3^{\binom{t}{2}})$. It is easy to prove (by induction on t) that

$$s_t \geq (7/3)^{-2/7} 9^{-2t/7} {s_0}^{\alpha(t)}.$$

Therefore

$$s_T \geq (7/3)^{-2/7} 9^{-2T/7} 2^{\alpha(T) \log n/5}.$$

To see that the claim follows, note that $3^{\binom{T}{2}} \leq (\log n)^{(\log 3)/4}$ and $(\log 3)/4 < 2/3$. □

FACT 3.5. *For large enough n and $t < T$, $3k_t \leq w_t^{1/7}$.*

Proof. Using Fact 3.4, for large enough n and $t < T$ we have

$$3k_t = k_{t+1} \leq 3^{\sqrt{\log \log n}} \leq 2^{\log^{1/3} n} \leq s_{t+1} = w_t^{1/7}. \qquad \square$$

FACT 3.6. $r_t / w_t^{4/7} > s_t^3$.

Proof. The proof is immediate from the definitions. □

DEFINITION. *A* t-good *partial h-relation is a partial h-relation f which satisfies the following three conditions.*

1. *r_t ranges have s_t processors that are mapped to "$*$" by f, and no processors that are mapped to "1" by f, while the remaining ranges have no processors mapped to "$*$" by f, and two processors that are mapped to "1" by f.*

2. *The (t, f)-knowledge set of each processor p has size at most one.*
3. *Each processor q is in the (t, f)-knowledge set of at most k_t processors.*

Condition 2 captures a crucial idea, which can be traced to Fich, Meyer auf der Heide, and Wigderson [8], and may be expressed informally as follows. Suppose that A is run on input g, where g is a 2-relation that refines f. Then the entire state of the n-OCPC at time t depends in a particularly simple way on the restriction of g to the processors p with $f(p) =$ "$*$."

3.4. Refining partial 2-relations with CONSTRUCT. At the heart of our proof is a randomized procedure CONSTRUCT(t, f) that takes a time t and a partial 2-relation f and returns a new partial 2-relation f' that is a refinement of f. Aside from the parameters t and f, CONSTRUCT depends implicitly on the algorithm A, in particular on the action of A at time step $t + 1$. (The approach here is similar to that used by MacKenzie in the context of lower bounds for load balancing [18].) The procedure CONSTRUCT has two important properties, the first of which is concerned with invariance. Namely, we will show that if $t < T$ and CONSTRUCT is called with parameters (t, f), where f is t-good, then with high probability, CONSTRUCT will return a partial 2-relation f' that is $(t + 1)$-good. The second property is that CONSTRUCT is unbiased. Specifically, suppose that GENERATE is a procedure that starts with the partial 2-relation $f_0 = f_*$ and applies CONSTRUCT T times to generate a sequence of partial relevant 2-relations $f_0 = f_* \geq f_1 \geq \cdots \geq f_T \geq f$ in which each $f_t =$ CONSTRUCT(t, f_{t-1}) is a refinement of f_{t-1}, and f is a relevant 2-relation generated u.a.r. from the set of refinements of f_T. We will show that the relevant 2-relation f produced by GENERATE is uniformly distributed. From invariance, we will also be able to conclude that with high probability the partial 2-relation f_T is T-good.

Before describing algorithm CONSTRUCT, we note that the proof of Theorem 3.2 follows quickly from the properties of CONSTRUCT that we have described, provided we are prepared to set aside a minor technical complication, which is dealt with later in this section. With high probability, the partial 2-relation f_T produced by GENERATE has many ranges with no processors mapped to "1" by f_T. In these ranges the target processor has a (T, f_T)-knowledge set of size at most one; thus the target processor can have received at most one of the messages destined for it.

We now describe algorithm CONSTRUCT which is called with a time t and a partial 2-relation f, and which randomly refines f based on the action of algorithm A at step $t + 1$. Let the jth range be denoted by R_j and let S_j denote the set of processors in R_j that are mapped to "$*$" by f. Let J be the set of indices j such that $|S_j| > 0$.

DEFINITION. *A processor p* zero-affects *a processor q if there is a processor p' such that p is in the (t, f)-knowledge set of p', and for any relevant 2-relation g which refines f and has $g(p) = 0$: when A is run with input g, processor p' sends to q on step $t + 1$.*

The notion of p *one-affecting* processor q is defined analogously. Whenever it is the case that a processor p is zero-affected or one-affected by a processor q, there is a risk that the $(t + 1, f)$-knowledge set of p will grow to size greater than one. Recall that the aim of CONSTRUCT is to produce a refinement f' of f that is $(t + 1)$-good; in particular this entails arranging that the $(t + 1, f')$-knowledge set of p has size at most one. CONSTRUCT's strategy is to nominate, for each range R_j with $j \in J$, a certain subset of S_j. The subsets are chosen in such a way that each processor p is affected by at most one processor in the subsets. Then CONSTRUCT randomly

selects a refinement f' of f such that the undetermined part of f' lies precisely over the union of the subsets that were nominated.

We now describe CONSTRUCT in detail. Let W'_j be a subset of S_j which is as large as possible and has the property that if two processors p_1 and p_2 are in W'_j and zero-affect the same processor q, then two processors in $S_j - W'_j$ also zero-affect processor q. Let W''_j be a subset of W'_j which is as large as possible and has the property that if two processors p_1 and p_2 are in W''_j and one-affect the same processor q, then all processors in W''_j one-affect processor q.

For each processor p in range R_j we define the set AFFECTS(p) as follows.

1. If p is in the (t, f)-knowledge set of any processor q then put q in AFFECTS(p).
2. If p zero-affects any processor q and there are not two processors in $S_j - W'_j$ which zero-affect q then put q in AFFECTS(p).
   (The intuition here is that if there are two processors in $S_j - W'_j$ which zero-affect q and all of the processors in $S_j - W'_j$ are mapped to "0" there will be a collision at processor q at step $t + 1$ so q will not be affected by p.)
3. If p one-affects any processor q and there is some processor in W''_j which does not one-affect q then put q in AFFECTS(p).
   (The intuition here is that if every processor in W''_j one-affects q and all of the processors in $S_j - W''_j$ are mapped to "0" there will be a collision at processor q at step $t + 1$ so q will not be affected by p.)

Let W_j be a subset of W''_j which is as large as possible and has the property that for any two processors p_1 and p_2 in W_j, AFFECTS(p_1) $\cap$ AFFECTS(p_2) is empty. (Intuitively, at this point, we would like each processor to be affected by at most one processor in each W_j.)

In CONSTRUCT, we will split J into groups $J_1, J_2, \ldots, J_\ell$ each of size $r_t/w_t^{4/7}$, with the last group possibly smaller. For each group J_i CONSTRUCT will construct a set V_i containing some of the processors from up to one of the ranges in J_i. The sets will have the property that if two processors p and p' are in $\bigcup_i V_i$, then AFFECTS(p)$\cap$ AFFECTS(p') is empty. Intuitively, this means that no processor could be affected by two processors in $\bigcup_i V_i$. We will let V denote $\bigcup_i V_i$. CONSTRUCT will produce f' by making random assignments to the processors which are not in V. We will say that algorithm CONSTRUCT is *successful* if each set V_i has size $w_t^{1/7}$.

```
Function CONSTRUCT(t, f)
    For each i ∈ {1, ..., ℓ}
        Let V_i = ∅
        For each j ∈ J_i
            Let S = ∅
            Let S' = ∅
            While |S| < w_t^{1/7} and |W_j − S − S'| > 0
                Let p be the lowest numbered processor in W_j − S − S'
                If there is no p' ∈ V_1 ∪ ··· ∪ V_{i−1} such that
                    AFFECTS(p) ∩ AFFECTS(p') ≠ ∅ Then
                    Let S = S ∪ {p}
                Else
                    Let S' = S' ∪ {p}
            Let f = RANDOMSET(S_j − S, f)
            If f maps any processor in S_j − S to "1" Then
                Let f = RANDOMSET(S, f)
                Next j
```

```
            Else
                Let V_i = S
                For each remaining j' ∈ J_i
                    Let f = RANDOMSET(S_{j'}, f)
                Next i
        Let f' = f
        Return f'
    End CONSTRUCT
```

3.5. Analysis of CONSTRUCT.

CLAIM 3.7. *If f is t-good then $|\text{AFFECTS}(p)| \leq 3k_t$ for each p.*

Proof. Since f is t-good, each p is in the (t, f) knowledge set of at most k_t processors. Each of these k_t processors can cause p to zero-affect at most one other processor and to one-affect at most one other processor. □

CLAIM 3.8. *If f is t-good then each processor q is in at most 3 sets* AFFECTS(p) *with $p \in W_j''$.*

Proof. Since f is t-good, the (t, f)-knowledge set of q has size at most one. Therefore, q is added to at most one set AFFECTS(p_1) using the first part of the definition of AFFECTS(p). By the construction of W_j', q is added to at most one set AFFECTS(p_2) using the second part of the definition of AFFECTS(p). Finally, by the construction of W_j'', q is added to at most one set AFFECTS(p_3) using the third part of the definition of AFFECTS(p). □

CLAIM 3.9. *If f is t-good then for each $j \in J$ we have $|W_j'| \geq |S_j|/(2k_t + 1)$.*

Proof. We use the following procedure, which we call Procedure A.

```
Procedure A
    For each j ∈ J
        Let S' = ∅
        Let S = S_j
        While |S| > 0
        Select a processor p ∈ S
            Let S = S − p
            Let S' = S' ∪ {p}
            For each processor q which p zero-affects
                Let Z = {v | v zero-affects q and v ∈ S}
                If Z > 1 Then
                    Let p_1, p_2 be two processors in Z
                    Let S = S − {p_1, p_2}
                Else
                    If Z = 1 Then
                        Let p_1 be the processor in Z
                        Let S = S − {p_1}
End A
```

Using Procedure A we can construct a set $S' \subseteq S_j$ such that if two processors p_1 and p_2 are in S' and zero-affect the same processor q, then two processors in $S_j - S'$ also zero-affect processor q. Procedure A starts by setting $S = S_j$. Since f is t-good each processor $p \in S$ zero-affects at most k_t processors. So for each iteration of the while loop at most $2k_t + 1$ processors are removed from S with exactly one of them placed in S'. Thus $|S'| \geq |S_j|/(2k_t + 1)$. By the definition of W_j', we have $|W_j'| \geq |S'| \geq |S_j|/(2k_t + 1)$. □

CLAIM 3.10. *If f is t-good then for each $j \in J$ we have $|W_j''| \geq |W_j'|^{1/k_t}/k_t$.*

Proof. For $p \in W'_j$, let $D(p)$ be the set of processors which p one-affects. Then $|D(p)| \leq k_t$. A *sunflower* is defined as a collection of sets such that if an element is in two of the sets, then it is contained in all of the sets. The Erdös–Rado theorem ([4]; see also [17]) says the following: let t and m be positive integers and let F be a family of sets such that every element of F has size at most t and $|F| > t!(m-1)^t$. Then F contains a sunflower of size m. If we let F be the family of sets $D(p)$ for $p \in W'_j$, then F contains a sunflower of size $(|W'_j|/k_t!)^{1/k_t} \geq |W'_j|^{1/k_t}/k_t$. If two processors p_1 and p_2 correspond to two sets in this sunflower and they one-affect the same processor q, then (by the definition of *D(p)* and sunflower) all p corresponding to sets in this sunflower one-affect q, and since W''_j is the largest set of processors which satisfy this property, $|W''_j| \geq |W'_j|^{1/k_t}/k_t$. □

A construction similar to the one used in the proof of Claim 3.10 was used by Grolmusz and Ragde [14].

CLAIM 3.11. *If f is t-good then for each $j \in J$ we have $|W_j| \geq |W''_j|/7k_t$.*

Proof. Construct a graph $G = (W''_j, E)$ where $(p,q) \in E$ if AFFECTS(p) ∩ AFFECTS(q) is nonempty. Then an independent set S in this graph has the property that for p_1, p_2 in S, AFFECTS(p_1) ∩ AFFECTS(p_2) is empty. Then W_j is simply the largest independent set in this graph. By Turán's theorem, $|W_j| \geq |W''_j|^2/(|W''_j| + 2|E|)$. By Claims 3.7 and 3.8, for each $p \in W''_j$, $|\text{AFFECTS}(p)| \leq 3k_t$, and each q is in at most 3 sets AFFECTS(p). Thus each $p \in W''_j$ is an endpoint of at most $6k_t$ edges in E and therefore $|E| \leq 3k_t\,|W''_j|$. We conclude that $|W_j| \geq |W''_j|/7k_t$. □

A construction similar to the one used in the proof of Claim 3.11 was used by Fich, Meyer auf der Heide, and Wigderson [8].

COROLLARY 3.12. *If f is t-good then for each $j \in J$ we have $|W_j| \geq w_t$.*

Proof. Since f is t-good, $|S_j| = s_t$. Then the corollary follows from Claims 3.9, 3.10, and 3.11. □

CLAIM 3.13. *If f is t-good then the number of groups used by algorithm* CONSTRUCT *is $w_t^{4/7}$.*

Proof. This follows from the definition of t-good and from the fact that the size of the groups is $r_t/w_t^{4/7}$. □

CLAIM 3.14. *If f is t-good and $t < T$ then the while loop in algorithm* CONSTRUCT *always terminates with $|S| = w_t^{1/7}$.*

Proof. We will show that if f is t-good then $|S| < w_t^{1/7}$ implies $|W_j - S - S'| > 0$. Suppose that some vertex p in W_j cannot be added to S. Then for some $p' \in V_1 \cup \cdots \cup V_{i-1}$ we have AFFECTS(p) ∩ AFFECTS(p') ≠ ∅. But the size of each set V_α is at most $w_t^{1/7}$ and i is at most the number of groups, which is equal to $w_t^{4/7}$ by Claim 3.13. Furthermore, for each $p' \in V_1 \cup \cdots \cup V_{i-1}$, $|\text{AFFECTS}(p')| \leq 3k_t$. So at most $3k_t w_t^{5/7}$ members of R_j will be put in S'. By Fact 3.5, $3k_t w_t^{5/7} < w_t - w_t^{1/7}$ for $t < T$ and large enough n. We conclude using Corollary 3.12 that if $|S| < w_t^{1/7}$ then $|W_j - S - S'| > w_t - (w_t^{1/7}) - (w_t - w_t^{1/7}) = 0$. □

CLAIM 3.15. *If f is t-good and $t < T$ then the probability that* CONSTRUCT *is successful is at least $1 - n^{-2}$.*

Proof. We have already shown in the proof of Claim 3.14 that if f is t-good and $t < T$ then the while loop in algorithm CONSTRUCT always terminates with $|S| = w_t^{1/7}$. It remains to show that with probability at least $1 - n^{-2}$ each group i has a range j such that the function f returned by the call "Let f = RANDOMSET($S_j - S, f$)" does not map any processor in $S_j - S$ to "1." Assume that this is true for groups 1 to $i-1$. For $1 \leq v \leq i-1$, let X_v be the random variable equal to the index of the

first such range in group v. For $1 \le j \le r_t/w_t^{4/7}$, let $Y_{i,j}$ be a binary random variable which is 1 when range j is such a range for group i. Let $Z_i = \sum_{j=1}^{r_t/w_t^{4/7}} Y_{i,j}$. Note that Z_i is zero if and only if group i does not have such a range. Note that for $j \neq j'$, $Y_{i,j}$ and $Y_{i,j'}$ are independent. By construction, for any $b_1, \ldots, b_{i-1} \in [1, r_t/w_t^{4/7}]$, using the facts that $s_t \ge 2^{\log^{1/3} n}$ (from Fact 3.4), and $r_t/w_t^{4/7} > s_t^3$ (from Fact 3.6), and assuming n is large,

$$
\begin{aligned}
&\Pr(Z_i = 0 | X_{i-1} = b_{i-1}, \ldots, X_1 = b_1) \\
&\quad = \Pr\left(\sum_{j=1}^{r_t/w_t^{4/7}} Y_{i,j} = 0 | X_{i-1} = b_{i-1}, \ldots, X_1 = b_1 \right) \\
&\quad = \Pr\left(\bigcap_{j=1}^{r_t/w_t^{4/7}} (Y_{i,j} = 0) | X_{i-1} = b_{i-1}, \ldots, X_1 = b_1 \right) \\
&\quad = \prod_{j=1}^{r_t/w_t^{4/7}} \Pr(Y_{i,j} = 0 | X_{i-1} = b_{i-1}, \ldots, X_1 = b_1) \\
&\quad = \left(1 - \frac{\binom{w_t^{1/7}}{2}}{\binom{s_t}{2}} \right)^{r_t/w_t^{4/7}} \\
&\quad \le \left(1 - \frac{1}{s_t^2} \right)^{s_t^3} \\
&\quad \le e^{-s_t} \\
&\quad \le n^{-3}.
\end{aligned}
$$

The probability of failing in any group can then be bounded by

$$
\begin{aligned}
&\sum_{i=1}^{w_t^{4/7}} \Pr(Z_i = 0 | Z_{i-1} = 1, \ldots, Z_1 = 1) \\
&\quad = \sum_{i=1}^{w_t^{4/7}} \sum_{b_1, \ldots, b_{i-1} \in [1, r_t/w_t^{4/7}]} \Pr(Z_i = 0 | X_{i-1} = b_{i-1}, \ldots, X_1 = b_1) \\
&\qquad\qquad \Pr(X_{i-1} = b_{i-1}, \ldots, X_1 = b_1) \\
&\quad \le n^{-3} \sum_{i=1}^{w_t^{4/7}} \sum_{b_1, \ldots, b_{i-1} \in [1, r_t/w_t^{4/7}]} \Pr(X_{i-1} = b_{i-1}, \ldots, X_1 = b_1) \\
&\quad = w_t^{4/7} n^{-3} \\
&\quad \le n^{-2}. \qquad \square
\end{aligned}
$$

COROLLARY 3.16. *If f is t-good and algorithm* CONSTRUCT *is successful then after* CONSTRUCT *is executed r_{t+1} ranges have s_{t+1} processors that are mapped to "$*$" by f', and no processors that are mapped to "1" by f', while the remaining ranges*

have no processors mapped to "$*$" *by* f'*, and two processors that are mapped to* "1" *by* f'

Proof. The proof is immediate from the definition of successful and from Claim 3.13. □

CLAIM 3.17. *If* f *is* t*-good then after* CONSTRUCT *is executed every processor* q *that is in the* $(t+1, f')$*-knowledge set of a processor* p *has* $p \in \text{AFFECTS}(q)$.

Proof. By the definition of dependency sets, we can form a $(t+1, f')$ dependency set D of p by taking the union of the (t, f)-knowledge set of p and the (t, f)-knowledge sets of all processors p' satisfying the following: there is some refinement g of f which is a relevant 2-relation and on which p' sends to p on step $t+1$. Note that D is the union of the (t, f)-knowledge set of p and the set of processors that zero-affect p and the set of processors that one-affect p. If q is in the (t, f)-knowledge set of p then p is in AFFECTS(q) by the first part of the definition of AFFECTS. Suppose that q_1 is a processor in some range j which zero-affects p and that $p \notin \text{AFFECTS}(q_1)$. By the second part of the definition of AFFECTS we know that there are two processors in $S_j - W'_j$ which zero-affect p. If both of these are mapped to "0" by f' then for any refinement of f' processor p has a conflict at step $t+1$ so $D - q_1$ is a $(t+1, f')$-dependency set of p. If, on the other hand, one of these is mapped to "1" by f' then algorithm CONSTRUCT maps every member of the range of q_1 to "0" or "1" so $D - q_1$ is a $(t+1, f')$-dependency set of p. (Recall that if $f'(q_1) \neq$ "$*$" then q_1 cannot be in the $(t+1, f')$-knowledge set of any processor.) Similarly, suppose that q_2 is a processor in some range j which one-affects p and that $p \notin \text{AFFECTS}(q_2)$. By the third part of the definition of AFFECTS we know that every processor in W''_j one-affects p. If all of the processors in $S_j - W''_j$ are mapped to "0" by f' then for any refinement of f' that is a relevant 2-relation processor p has a conflict at step $t+1$ so $D - q_2$ is a $(t+1, f')$-dependency set of p. If, on the other hand, one of these is mapped to "1" by f' then algorithm CONSTRUCT maps every member of the range of q_2 to "0" or "1" so $D - q_2$ is a $(t+1, f')$-dependency set of p. □

CLAIM 3.18. *If* f *is* t*-good and algorithm* CONSTRUCT *is successful then after* CONSTRUCT *is executed the* $(t+1, f')$*-knowledge set of every processor* p *has size at most one.*

Proof. We know from Claim 3.17 that every processor p has a $(t+1, f')$-dependency set D which contains only those processors q such that $p \in \text{AFFECTS}(q)$. Suppose that two processors q and q' have $f'(q) = f'(q') =$ "$*$." (If a processor q is not mapped to "$*$" by f' then it is not in the $(t+1, f')$-knowledge set of any processor so it is not in the $(t+1, f')$-knowledge set of p.) Then q must be in some $W_j \subset W''_j \subset W'_j$ and q' must be in some $W_{j'} \subset W''_{j'} \subset W'_{j'}$ and both q and q' are in the set V constructed by algorithm CONSTRUCT. If $j = j'$, then the definition of W_j guarantees that $\text{AFFECTS}(q) \cap \text{AFFECTS}(q') = \emptyset$, implying that p is in just one of these sets, and thus either q or q' is not in D. If, on the other hand, $j \neq j'$ by the construction of V, $\text{AFFECTS}(q) \cap \text{AFFECTS}(q') = \emptyset$, implying p is in just one of these sets, and thus either q or q' is not in D. Thus $|D| \leq 1$. □

CLAIM 3.19. *If* f *is* t*-good then after* CONSTRUCT *is executed each processor* q *is in the* $(t+1, f')$*-knowledge set of at most* k_{t+1} *processors.*

Proof. Let q be a processor which is in the $(t+1, f')$-knowledge set of a processor p. By Claim 3.17, $p \in \text{AFFECTS}(q)$. But by Claim 3.7, $|\text{AFFECTS}(q)| \leq 3k_t = k_{t+1}$. The claim follows. □

LEMMA 3.20. *If* $t < T$ *and* CONSTRUCT *is called with parameters* (t, f)*, where* f *is* t*-good, then with probability at least* $1 - n^{-2}$ CONSTRUCT *will return a partial*

2-*relation* f' *that is* $(t+1)$-*good.*

Proof. The proof follows from Claims 3.15, 3.18, and 3.19 and Corollary 3.16. □

3.6. Function GENERATE. We use the following function, which calls CONSTRUCT to generate a sequence of partial relevant 2-relations $f_0 = f_* \geq f_1 \geq \cdots \geq f_T \geq f$ in which each f_t is a refinement of f_{t-1}, f is a refinement of f_T, and f is a relevant 2-relation generated u.a.r.

```
Function GENERATE
    Let f_0 = f_*
    Let f = f_0
    Let t = 0
    While t ≤ T Do
        If for some p, f(p) = "*" Then
            Let f_t = CONSTRUCT(t, f)
        Else
            Let f_t = f
        t = t + 1
        f = f_t
    Let P = {p | f(p) = "*"}
    Return RANDOMSET(f, P)
End GENERATE
```

LEMMA 3.21. *The relevant* 2-*relation* f *produced by* GENERATE *is uniformly distributed.*

Proof. The lemma follows from Claim 3.3. □

LEMMA 3.22. *With probability at least* $1 - n^{-1}$, *the partial* 2-*relation* f_T *is* T-*good.*

Proof. Let Z_t be a random variable which is equal to 1 when CONSTRUCT succeeds at step t. Then the probability of failing at any step $t \leq T$ can then be bounded by

$$\sum_{t=1}^{T} \Pr(Z_t = 0 \mid Z_{t-1} = 1, \ldots, Z_1 = 1).$$

By Lemma 3.20, this is at most Tn^{-2} which is at most n^{-1}. □

We now prove the following theorem.

THEOREM 3.2. *Let* A *be a deterministic algorithm that allegedly routes* 2-*relations in* $T = \sqrt{\log\log n/2}$ *steps. Let the input to* A *be drawn u.a.r. from the set of relevant* 2-*relations. Then the probability that* A *successfully routes the input is at most* $\frac{1}{2}$.

Proof. We will generate a relevant 2-relation by running algorithm GENERATE. By Lemma 3.21, algorithm GENERATE generates relevant 2-relations u.a.r. GENERATE also produces a sequence $f_0 \geq \cdots f_T \geq \cdots f$ in which f is the final relevant 2-relation. By Lemma 3.22, f_T will be T-good with probability at least $1 - 1/n$.

Suppose that f_T is T-good. Then there is a range R that has a set S of s_T processors which are mapped to "$*$" by f_T. R has no processors which are mapped to "1" by f_T. Let d denote the first processor in range R. (d is the destination of the messages in range R.) The (T, f_T)-knowledge set of d contains at most one processor. There are three cases which must be examined concerning f_T.

Case 1. The (T, f_T)-knowledge set of d contains a processor q which is a member of S: we wish to bound the probability that A succeeds, given that f_T is in Case 1. Let $\mathcal{F}_1$ denote the set of relevant 2-relations which refine f_T and map q to "1" and

let $\mathcal{F}_0$ denote the set of relevant 2-relations which refine f_T and map q to "0." One can see by examining algorithm RANDOMSET that the probability that f is in $\mathcal{F}_1$ is $2/s_T$ and the probability that f is in $\mathcal{F}_0$ is $1 - 2/s_T$. We now examine the following subcases concerning f.

Case 1A. f is in $\mathcal{F}_1$: we wish to bound the probability that A succeeds, given that f is in $\mathcal{F}_1$. There is a particular trace τ which is the (T, f')-trace of d for every input h-relation $f' \in \mathcal{F}_1$. Since A runs in T steps processor d uses this trace τ to deduce the pair of messages that were destined for d in every input h-relation that is in $\mathcal{F}_1$. But there are $s_T - 1$ such pairs of messages, each of which is equally likely to come up in a randomly chosen member of $\mathcal{F}_1$. So the probability that A is successful given that f is in $\mathcal{F}_1$ is at most $1/(s_T - 1)$.

Case 1B. f is in $\mathcal{F}_0$: we wish to bound the probability that A succeeds, given that f is in $\mathcal{F}_0$. There is a particular trace τ which is the (T, f')-trace of d for every input h-relation $f' \in \mathcal{F}_0$. Since A runs in T steps processor d uses this trace τ to deduce the pair of messages that were destined for d in every input h-relation that is in $\mathcal{F}_0$. But there are $\binom{s_T-1}{2}$ such pairs of messages, each of which is equally likely to come up in a randomly chosen member of $\mathcal{F}_1$. So the probability that A is successful given that f is in $\mathcal{F}_1$ is at most $1/\binom{s_T-1}{2}$.

Therefore the probability that A succeeds given that f_T is in Case 1 is at most $(2/s_T)(1/(s_T - 1)) + (1 - 2/s_T)(1/\binom{s_T-1}{2})$ which is at most $2/\binom{s_T-1}{2}$.

Case 2. The (T, f_T)-knowledge set of d contains a processor q which is not a member of S: similar arguments to those used in Case 1 show that the probability that A succeeds given that f_T is in Case 2 is at most $1/\binom{s_T}{2}$.

Case 3. The (T, f_T)-knowledge set of d is the empty set: similar arguments to those used in Case 1 show that the probability that A succeeds given that f_T is in Case 3 is at most $1/\binom{s_T}{2}$.

Finally, we conclude that the probability that A successfully routes f in T steps is at most the sum of $1/n$ (an upper bound on the probability that f_T is not T-good, by Lemma 3.22) and $(1 - 1/n) \times 2/\binom{s_T-1}{2}$ (an upper bound on the probability that A succeeds given that f_T is T-good). We can use Fact 3.4 to show that this quantity is at most $1/2$.

Therefore, with probability at least $1/2$, an f drawn u.a.r. from the set of relevant 2-relations will not be routed by algorithm A in T steps. □

COROLLARY 3.23. *Let A be a deterministic algorithm that routes 2-relations. Let the input to A be drawn u.a.r. from the set of relevant 2-relations. Then the expected number of communication steps used by A is at least $\sqrt{\log \log n}/4$.*

Proof. The corollary follows from the fact that $\sqrt{\log \log n}/4 \leq (1/2)(T+1)$. □

The following weak form of a theorem of Yao is stated (and proved) in Fich, Ragde, and Wigderson's paper [9].

THEOREM 3.24 (see [23]). *Let T_1 be the expected running time for a given probabilistic algorithm solving problem P, maximized over all possible inputs. Let T_2 be the average running time for a given input distribution, minimized over all possible deterministic algorithms to solve P. Then $T_1 \geq T_2$.* □

We now prove the following theorem.

THEOREM 3.1. *Let A be a randomized algorithm that routes 2-relations on an n-OCPC. Then there is a 2-relation on which the expected number of communication steps used by A is at least $\sqrt{\log \log n}/4$.*

Proof. Corollary 3.23 shows that the average running time for the uniform distribution on relevant 2-relations, minimized over all deterministic algorithms, is at least

$\sqrt{\log\log n}/4$. Theorem 3.1 now follows from Theorem 3.24. □

4. The prospect for tightening the bound. Recall the situation in which two processors p and q each have a single message to transmit to a common destination. Consider the following OCPC "algorithm" which is a parallel version of a strategy consider in section 2. In $\Theta(\sqrt{\log\log n})$ steps, p and q recruit $k = \Theta\big(\exp(\sqrt{\log\log n}\,)\big)$ "agents" to help discover a bit position at which the binary sequence numbers for p and q differ. This is done using the method of section 2 but with k-way search in place of binary search: a p-agent and a q-agent simultaneously attempt to transmit a message to processors with sequence numbers of the form $0\ldots0p_{i+1}\ldots p_{i+r/k}0\ldots0$ and $0\ldots0q_{i+1}\ldots q_{i+r/k}0\ldots0$, respectively, and hence discover whether the sequence numbers of p and q differ on a particular block of r/k bits. This would seem to give an $O(\sqrt{\log\log n}\,)$ algorithm for delivering the messages.

Of course, the catch is that a p-agent that finds a block on which the sequence numbers of p and q differ is unable to alert the other p-agents to the discovery, at least not sufficiently quickly to obtain an improvement over the original binary search strategy. Unfortunately, the lower bound argument presented here is oblivious to a cheating "algorithm" in which an agent that finds an appropriate block broadcasts its discovery to the other agents in one step. The problem is that in the lower bound argument, the behavior of a processor is considered to be a function of a partial 2-relation f that provides far more information than a processor could in reality know.

Acknowledgments. The authors thank Satish Rao and Mike Sipser for useful discussions.

REFERENCES

[1] R. J. Anderson and G. L. Miller, *Optical Communication for Pointer Based Algorithms*, Technical Report CRI 88-14, Computer Science Department, University of Southern California, Los Angeles, CA, 1988.

[2] M. Dietzfelbinger and F. Meyer auf der Heide, *Simple Efficient Shared Memory Simulations*, in Proceedings of the ACM Symposium on Parallel Algorithms and Architectures 5, 1993, pp. 110–119.

[3] P. W. Dowd, *High performance interprocessor communication through optical wavelength division multiple access channels*, Symp. on Computer Architectures 18, 1991, pp. 96–105.

[4] P. Erdős and R. Rado, *Intersection theorems for systems of sets*, J. London Math. Soc., 35 (1960), pp. 85–90.

[5] M. M. Eshaghian, *Parallel Computing with Optical Interconnects*, Ph.D. thesis, University of Southern California, Los Angeles, CA, Dec 1988.

[6] M. M. Eshaghian, *Parallel Algorithms for Image Processing on OMC*, IEEE Trans. Comput., 40 (1991), pp. 827–833.

[7] M. M. Eshaghian and V. K. P. Kumar, *Optical arrays for parallel processing*, in Proceedings of the Second Annual Parallel Processing Symposium, 1988, pp. 58–71.

[8] F. E. Fich, F. Meyer auf der Heide, and A. Wigderson, *Lower bounds for parallel random-access machines with unbounded shared memory*, Advances in Computing Research 4, F. Preparata, Ed., JAI Press, Greenwich, CT, 1987, pp. 1–15.

[9] F. E. Fich, P. Ragde, and A. Wigderson, *Relations between concurrent-write models of parallel computation*, SIAM J. Comput., 17 (1988), pp. 606–627.

[10] M. Furst, J. B. Saxe, and M. Sipser, *Parity, circuits, and the polynomial time hierarchy*, Math. Systems Theory, 17 (1984), pp. 13–28.

[11] A. V. Gerbessiotis and L. G. Valiant, *Direct Bulk-Synchronous Parallel Algorithms*, Scandinavian Workshop on Algorithm Theory 3, Springer-Verlag, Berlin, New York, 1992.

[12] M. Geréb-Graus and T. Tsantilas, *Efficient optical communication in parallel computers*, in Proceedings of the ACM Symposium on Parallel Algorithms and Architectures 4, 1992, pp. 41–48.

[13] L. A. Goldberg, M. Jerrum, T. Leighton, and S. Rao, *A doubly logarithmic communication algorithm for the completely connected optical communication parallel computer*, in

Proceedings of the ACM Symposium on Parallel Algorithms and Architectures 5, 1993, pp. 300–309.

[14] V. Grolmusz and P. Ragde, *Incomparability in parallel computation*, in Proceedings of the IEEE Symposium on Foundations of Computer Science 28, 1987, pp. 89–98.

[15] J. Hastad, *Computational Limitations for Small Depth Circuits*, MIT Press, Cambridge, MA, 1987.

[16] R. Impagliazzo, R. Paturi, and M. E. Saks, *Size-depth trade-offs for threshold circuits*, in Proceedings of the ACM Symposium on Theory of Computing 25, 1993, pp. 541–550.

[17] M. Luby and B. Veličković, *On deterministic approximation of DNF*, in Proceedings of the ACM Symposium on Theory of Computing 23, 1991, pp. 430–438.

[18] P. D. MacKenzie, *Load balancing requires* $\Omega(\log^* n)$ *expected time*, in Proceedings of the ACM-SIAM Symposium On Discrete Algorithms 3, 1992, pp. 94–99.

[19] P. D. MacKenzie, C. G. Plaxton, and R. Rajaraman, *On contention resolution protocols and associated probabilistic phenomena*, in Proceedings of the ACM Symposium on Theory of Computing 26, 1994, pp. 153–162.

[20] P. D. MacKenzie and V. Ramachandran, *ERCW PRAMs and optical communication*, in Proceedings Euro-Par '96, Springer-Verlag LNCS 1124, Lyon, France, August 1996, pp. 293–302.

[21] S. B. Rao, *Properties of an Interconnection Architecture Based on Wavelength Division Multiplexing*, Technical Report TR-92-009-3–0054-2, NEC Research Institute, 4 Independence Way, Princeton, NJ 08540, 1992.

[22] L. G. Valiant, *General purpose parallel architectures*, Chapter 18 of *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed., Elsevier, New York, 1990; see especially p. 967.

[23] A. C.-C. Yao, *Probabilistic computations: Toward a unified measure of complexity*, in Proceedings of the 18th Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1977, pp. 222–227.

SIAM J. COMPUT.
Vol. 27, No. 4, pp. 1099–1115, August 1998
© 1998 Society for Industrial and Applied Mathematics
010

COMPUTING MATRIX EIGENVALUES AND POLYNOMIAL ZEROS WHERE THE OUTPUT IS REAL*

DARIO BINI† AND VICTOR Y. PAN‡

Abstract. Surprisingly simple corollaries from the Courant–Fischer minimax characterization theorem enable us to devise a very effective algorithm for the evaluation of a set S interleaving the set E of the eigenvalues of an $n \times n$ real symmetric tridiagonal (rst) matrix T_n (as well as a point that splits E into two subsets of comparable cardinalities). Furthermore, we extend this algorithm so as to approximate all the n eigenvalues of T_n at *nearly optimal* sequential and parallel cost, that is, at the cost of staying within polylogarithmic factors from the straightforward lower bounds. The resulting improvement of the known processor bound in NC algorithms for the rst-eigenproblem is roughly by factor n. Our approach extends the previous works [M. Ben-Or and P. Tiwari, *J. Complexity*, 6(1990), pp. 417–442] and [M. Ben-Or et al., *SIAM J. Comput.*, 17(1988), pp. 1081–1092] for the approximation of the zeros of a polynomial having only real zeros, and our algorithm leads to an alternative and simplified derivation of the known record parallel and sequential complexity estimates for the latter problem.

Key words. symmetric tridiagonal eigenvalues, approximation algorithms, computational complexity, real polynomial zeros

AMS subject classifications. 68Q05, 68Q40, 68H05

PII. S0097539790182482

1. Introduction.

1.1. The problem. The problem of approximating the eigenvalues of an $n \times n$ Hermitian or real symmetric matrix A is one of the central problems of practical matrix computations [GL], [Par]. The first step of its solution in all the customary algorithms is the reduction of the input matrix A to the real symmetric tridiagonal (rst) form; this step can be effectively parallelized [Pan87], [BP94, Proposition 5.4, p. 325].

In the present paper, we consider the remaining eigenvalue problem for an $n \times n$ rst-matrix T_n. From technical and theoretical points of view, this problem is closely related to approximating the zeros of polynomials having only real zeros. In computing practice, such polynomials usually appear as the characteristic polynomials of Hermitian (or real symmetric) matrices, which also cover the class of orthogonal polynomials. Given the coefficients of an nth-degree polynomial $p(\lambda)$ having only real zeros, $\lambda_1, \ldots, \lambda_n$, we may compute the entries of an $n \times n$ rst-matrix T_n that has the characteristic polynomial $p(\lambda)$ and the eigenvalues $\lambda_1, \ldots, \lambda_n$. In Appendix A and also in [BP94, pp. 117–120], we achieve this by applying the extended Euclidean scheme to $p(\lambda)$ and $p'(\lambda)$, whereas, for a given T_n, we may compute the coefficients of $p(\lambda)$ by applying a simpler algorithm, which we present in section 5. In the practice of matrix computations, the reduction of the rst-eigenvalue problem to computing

* Received by the editors June 1, 1990; accepted for publication June 11, 1996; published electronically May 19, 1998. The results of this paper were presented at the Second Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, 1991.
http://www.siam.org/journals/sicomp/27-4/18248.html
† Dipartimento di Matematica, Università di Pisa, 56100 Pisa, Italy (bini@dm.unipi.it). This research was supported by NSF grant 8805782 and by the Italian M.P.I. 40% funds.
‡ Department of Mathematics and Computer Science, Lehman College, City University of New York, Bronx, NY 10468 (vpan@lcvax.lehman.cuny.edu). This research was supported by NSF grant 8805782 and by PSC-CUNY Awards 668541 and 669210.

the coefficients of the characteristic polynomial $p(\lambda)$ is avoided because of the numerical stability problems (in particular, the value $|p(0)| = |\det T_n|$ can be very large); this does not apply, however, to computing the *values* of $p(\lambda)$, which is a customary auxiliary step of the rst-eigenvalue computation [Par], [GL, pp. 437–440].

1.2. Our results (outline). Our main result is a new parallel NC algorithm for approximating the eigenvalues of an rst-matrix T_n given its entries. We assume the customary arithmetic and Boolean PRAM models of parallel computation (where in each parallel step each nonidle processor performs one arithmetic or, respectively, Boolean operation), and we deduce *nearly optimum* upper bounds on parallel time and the number of processors required by our algorithm. (By nearly optimum we mean upper bounds that are within polylogarithmic factors from the known, and in our case straightforward, lower bounds.) This is a dramatic improvement of the previous best parallel solution, based on the reduction of the problem to approximating the zeros of $p(\lambda)$ and on the solution of the latter problem by means of the algorithm of [BOT]. Our approach avoids using extended Euclidean computations involved in the algorithm of [BOT], and this gives us the edge over [BOT].

The sequential version of our algorithm, as well as its extension to the sequential and parallel approximation of the zeros of $p(\lambda)$ given the coefficients, supports the same complexity estimates as the algorithm of [BOT] does, and in the sequential case, these upper estimates are nearly optimal too. Our approach, however, may be considered conceptually simpler and easier to comprehend, as the reader may observe from the comparison of our techniques with ones of [BOT], made in the beginning of section 2.

1.3. Our complexity estimates. To estimate the sequential and parallel cost, we will write $O_A(t,p)$ and $O_B(t,p)$ for the algorithms that require $O(t)$ parallel steps using p arithmetic processors and $O(t)$ parallel steps using p bit-serial processors, respectively. The bounds $O_A(t,sp)$ and $O_B(t,sp)$ imply the bounds $O_A(st,p)$ and $O_B(st,p)$, respectively, for $s \geq 1$, according to Brent's scheduling principle of parallel computing [Br]. (In particular, $O_A(t,p)$ and $O_B(t,p)$ imply the sequential arithmetic cost bound $O_A(tp,1)$ and the sequential Boolean cost bound $O_B(tp,1)$, respectively.) Under this notation, our algorithm supports approximating the eigenvalues of T_n (whose entries have magnitudes at most 2^m), within the absolute error bound 2^{-h}, at the arithmetic and Boolean parallel cost bounded by $O_A(\log^2 n(\log^2 b + \log n)\log\log n, n/\log\log n)$ and $O_B(\log^2 n \log(nb)(\log^2 b + \log n)\log\log n \log\log(nb), n^2 b/\log\log n)$, respectively, and at the sequential cost bounded by $O_A(n\log^2 n(\log^2 b + \log n), 1)$ and $O_B(n^2 b \log^2 n \log(nb)(\log^2 b + \log n)\log\log(nb), 1)$, respectively, where $b = m + h$. For approximating all the n zeros of a polynomial $p(\lambda)$ having only real zeros, the same sequential cost bounds hold, but the parallel cost bounds change (in particular, to $O_A(\log^2 n(\log^2 b + \log n), (n/\log b)^2)$ under the arithmetic PRAM model) since we need to add the cost of performing the extended Euclidean algorithm that supports the transition from $p(\lambda)$ to T_n. In section 8, we comment on some ways to further minor improvements.

1.4. A parallel modification. Our major concern in this paper is about the computational complexity estimates. On the other hand, caring more about numerical stability of lower precision computations than about decreasing their asymptotic complexity, we have modified our main algorithm in [BP92] (cf. our Remark 5.1 in section 5 and comments in section 8). In the parallel NC algorithm of [BP92], we only need about $n/\log n$ times more processors but achieve substantially improved

numerical stability, which makes the algorithm competitive with the known alternative practical algorithms for the symmetric eigenvalue problem. In [B] and in [BG1], further progress in this direction has been obtained.

1.5. Organization of the paper. We will organize our paper as follows: in section 2, we deduce some properties of interlacing sets by using the Cauchy interlace theorem for the matrix eigenvalues. In section 3, for each eigenvalue of T_n, we compute either its approximation within a fixed error bound or an interval containing this eigenvalue but no other eigenvalues of T_n. In section 4, given an interval containing only one eigenvalue, we compute an approximation to this eigenvalue within the required precision by means of Newton's method and of the bisection of the exponents. In section 5, we describe an algorithm for the simultaneous computation of the values of the characteristic polynomial and its derivative at a set of points. In section 6, we summarize the construction of sections 2–5 and devise an algorithm for the approximation of all the eigenvalues. In section 7, we estimate the bit-complexity of the algorithm. In section 8, we briefly discuss some results that appeared more recently, after this paper had been submitted for publication—in particular, some results related to the complexity estimates and to practical performance of the algorithm of the present paper. In section A.1 of the appendix, we discuss the relations between the polynomial root-finding problem and the problem of the computation of the eigenvalues of a matrix, and in section A.2, the reduction of a Hermitian or real symmetric matrix to the tridiagonal form.

2. Interlacing sets and splitting points for the set of the eigenvalues. In the following, all logarithms are to the base 2.

Hereafter, T_n denotes an $n \times n$ rst-matrix having diagonal entries $a_1, \dots, a_n$, subdiagonal entries $b_1, \dots, b_{n-1}$, and a set Λ of n eigenvalues, $\Lambda = \{\lambda_1 \le \cdots \le \lambda_n\}$. Without loss of generality, suppose that $b_1 \dots b_{n-1} \neq 0$. (Indeed, if $b_j = 0$ for some j, the eigenvalue problem would be split into two eigenvalue problems of smaller dimensions.) We assume that $|a_i|$, $|b_i| \le 2^m$, m is a positive integer, and then, by virtue of Gerschgorin's theorem ([GL, p. 341]),

$$-3(2^m) \le \lambda_i \le 3(2^m). \tag{2.1}$$

We say that the set $R = \{r_0, \dots, r_k\}$ *interleaves* the set $Q = \{q_1, \dots, q_k\}$ and that R is an *interlacing set* for Q if $r_0 \le q_1 \le r_1 \le q_2 \le \cdots \le q_{k-1} \le r_{k-1} \le q_k \le r_k$, where, in particular, we will allow $r_0 = -\infty$ and/or $r_k = +\infty$, and then we will write $R = \{r_i, \dots, r_{k-j}\}$, where $i, j = 0, 1$. We say that s is a *splitting point of the level* (g, h) for the set Q if $q_g < s < q_h$.

In this section, we will prove some simple corollaries from Cauchy's interlace theorem [Par, p. 186] or from the Courant–Fischer minimax characterization [GL, p. 411]. Later on, they will lead us to simple algorithms for the evaluation of the interlacing sets and splitting points for the set of the eigenvalues of T_n. Using such sets and/or points will enable us to devise divide-and-conquer algorithms for approximating the eigenvalues of T_n, and we will specify such an algorithm in section 6.

It is instructive to compare these results with the techniques of [BOT] and [BFKT] dealing with a polynomial $p(\lambda)$ that has only real zeros and can be considered as the characteristic polynomial of T_n. Specifically, the algorithm of [BOT] relies on computing the Sturm and pseudoremainder sequences associated with $p(\lambda)$ and defining a set S of real points that interleaves the set of the zeros of $p(\lambda)$. Computing such a set S is the central (and also most innovative, most intricate, and most involved) part of

the algorithm of [BOT], and here we will introduce our main innovation too: we will replace the major techniques of [BOT] by some simple corollaries from Cauchy's interlace theorem or the Courant–Fischer minimax characterization theorem, and then, we will immediately arrive at a set interleaving the set of the eigenvalues Λ of T_n.

Technically, it may also be interesting that Cauchy's theorem or the Courant–Fischer minimax characterization may replace Sturm sequences in at least one more application. Namely, in [BFKT], the Sturm sequences have been used for computing a splitting point (rather than the interleaving set) for the set of the zeros of $p(\lambda)$; then again, some simple corollary from Cauchy's theorem or the Courant–Fischer minimax characterization will give us a simple algorithm for computing such a splitting point for the eigenvalues of T_n, and this computation is more effective than one of [BFKT]. We will give more comments later on, after the statement of our Corollary 2.1.

Hereafter, $\mathrm{Diag}(B_1, \dots, B_s)$ denotes the block diagonal matrix having diagonal blocks $B_1, \dots, B_s$.

We will rely on the following result, known as Cauchy's interlace theorem.

THEOREM 2.1. *If A_r denotes an $r \times r$ principal submatrix of an $n \times n$ real symmetric matrix A, then the eigenvalues $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_n$ of A and the eigenvalues $\mu_1 \le \mu_2 \le \cdots \le \mu_r$ of A_r satisfy the following relations:*

$$\lambda_i \le \mu_i \le \lambda_{i+n-r}, \quad i = 1, \dots, r.$$

Proof. The latter relations hold for $r = n-1$ (see Corollary 8.1-4 to the Courant–Fischer theorem in [GL, p. 411]). In the general case, it is sufficient to apply these relations to a sequence $A = A_n, A_{n-1}, A_{n-2}, \dots, A_r$ of principal submatrices of A such that A_i is a principal submatrix of A_{i+1}. An alternative proof can be found on pp. 186–187 of [Par]. □

We are ready to prove the following basic result.

THEOREM 2.2. *The eigenvalues of T_n satisfy the following relations:*

(a) *If nk is a multiple of $2(k+1)$ and if $\{\mu_1 < \mu_2 < \cdots < \mu_{\frac{k}{k+1}n}\}$ is the set of all the eigenvalues of the $k \times k$ principal submatrices $\tilde{T}_i$ of T_n, $i = 1, \dots, \frac{n}{k+1}$, $\tilde{T}_i$ containing the (s,s) entries a_s of T_n for $s = (i-1)(k+1)+j$, $j = 1, \dots, k$, then*

$$\lambda_{\frac{nk}{2(k+1)}} \le \mu_{\frac{nk}{2(k+1)}} \le \lambda_{\frac{n(k+2)}{2(k+1)}}.$$

(b) *If $1 \le j \le n-2$ and if $\{\gamma_1 \le \gamma_2 \le \cdots \le \gamma_{n-1}\}$ is the set of all the eigenvalues of the two principal submatrices of T_n,*

$$T_j = \begin{pmatrix} a_1 & b_1 & & \\ b_1 & a_2 & \ddots & \\ & \ddots & \ddots & b_{j-1} \\ & & b_{j-1} & a_j \end{pmatrix}, \quad \hat{T}_{n-j-1} = \begin{pmatrix} a_{j+2} & b_{j+2} & & \\ b_{j+2} & a_{j+3} & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & b_{n-1} & a_n \end{pmatrix},$$

then

$$\lambda_i \le \gamma_i \le \lambda_{i+1}, \quad i = 1, \dots, n-1.$$

In other words, part (a) *defines a splitting point of the level $(\frac{nk}{2(k+1)}, \frac{n(k+2)}{2(k+1)})$ for the set of the eigenvalues of T_n, and part* (b) *defines an interlacing set for this set of the eigenvalues.*

Proof. Theorem 2.2 immediately follows from Theorem 2.1. In particular, to prove part (a), apply Theorem 2.1 to the $r \times r$ principal submatrix of T_n (where $r = n - \frac{n}{k+1}$) obtained by deleting the $(i(k+1))$th rows and columns of T_n for $i = 1, 2, \ldots, \frac{n}{k+1}$. This submatrix is the block diagonal matrix $\mathrm{Diag}(\tilde{T}_1, \ldots, \tilde{T}_{\frac{n}{n+1}})$. To prove part (b), apply Theorem 2.1 to the $(n-1) \times (n-1)$ principal submatrix of T_n obtained by deleting the ith row and column of T_n; this submatrix is the 2×2 block diagonal matrix $\mathrm{Diag}(T_j, \hat{T}_{n-j-1})$. □

Remark 2.1. To extend part (a) to the case of any n, apply Theorem 2.2 to the tridiagonal matrix

$$\begin{pmatrix} T_n & O \\ O & qI_s \end{pmatrix}$$

for an appropriate $s < 2k+2$ and any fixed real q, where I_s is the $s \times s$ identity matrix.

Apply part (a) of Theorem 2.2 for $k = 1$, then modify it by replacing $\tilde{T}_i$ by the 1-by-1 matrices (a_{2i}) for $i = 1, \ldots, \frac{n}{2}$, and thus arrive at Corollary 2.1.

COROLLARY 2.1. *If n is a multiple of 4, $\{\sigma_1 \le \sigma_2 \le \cdots \le \sigma_{\frac{n}{2}}\} = \{a_{2i-1}, i = 1, \ldots, \frac{n}{2}\}$, $\{\theta_1 \le \theta_2 \le \cdots \le \theta_{\frac{n}{2}}\} = \{a_{2i}, i = 1, \ldots, \frac{n}{2}\}$, then*

$$\lambda_{\frac{n}{4}} \le \sigma_{\frac{n}{4}} \le \lambda_{\frac{3}{4}n}, \quad \lambda_{\frac{n}{4}} \le \theta_{\frac{n}{4}} \le \lambda_{\frac{3}{4}n}.$$

Corollary 2.1 in a weaker form has been proven in [BFKT], where it states some properties of the zeros of the polynomials generated by the Euclidean scheme for the two polynomials $p(\lambda) = \det(\lambda I - T_n)$ and $p'(\lambda)$ and where it is used in order to solve the root-finding problem in NC for a polynomial having only real zeros. The result of part (b) of Theorem 2.2 has been proven in [BOT], still in terms of the zeros of the polynomials generated by the Euclidean scheme. The proofs in both papers [BFKT] and [BOT] are quite intricate, whereas the matrix formulation of these properties is a straightforward consequence of Theorem 2.1, and the results can be immediately extended to polynomial zeros (as we pointed out in the introduction).

Let us again apply the Courant–Fischer theorem in order to obtain still another interlacing set for the set Λ of the eigenvalues of T_n; this time, we will rely on a suitable rank-one modification of the matrix T_n (cf. Remark 2.1 at the end of this section).

THEOREM 2.3. *Let $\{\phi_1 \le \phi_2 \le \cdots \le \phi_n\}$ be the set of all the eigenvalues of the matrices $S_k = T_k - \mathrm{Diag}(0, \ldots, 0, b_k)$, $R_{n-k} = \hat{T}_{n-k} - \mathrm{Diag}(b_k, \ldots, 0)$, where T_k and $\hat{T}_{n-k}$ are defined in Theorem 2.2. Set $\phi_{n+1} = \phi_n + 2b_k$, $\phi_0 = \phi_1 + 2b_k$.*

If $b_k > 0$, then

$$\phi_i \le \lambda_i \le \phi_{i+1}, \quad i = 1, \ldots, n.$$

If $b_k < 0$, then

$$\phi_{i-1} \le \lambda_i \le \phi_i, \quad i = 1, \ldots, n.$$

Proof. Theorem 2.3 follows from Theorem 8.1-5 of [GL, p. 412], applied to the matrix equation

$$T_n = \mathrm{Diag}(S_k, R_{n-k}) + 2b_k \mathbf{e}\mathbf{e}^T,$$

where $\mathbf{e} = (e_i)$, $e_i = 1/\sqrt{2}$ for $i = k$, and $i = k+1$, $e_i = 0$ elsewhere. □

A different proof of Theorem 2.3 is given in [Cu] and [BNS], where this theorem is used for separating the eigenvalues of an rst-matrix as a basis for devising practically effective divide-and-conquer algorithms for approximating the eigenvalues and eigenvectors of rst-matrices. As in [BNS] and [Cu], we may extend our algorithm to computing the eigenvectors of T_n, although our approach does not require us to compute the eigenvectors if we only need to compute the eigenvalues.

Remark 2.2. The algorithm supporting our asymptotic complexity estimates can be based on Theorems 2.2 or 2.3 as well. Application of Theorem 2.3 leads to some advantages for practical computation, particularly due to the possibility of using the so-called secular function (cf. [Cu], [BP92]).

3. Computing the number of the eigenvalues in the intervals of nearly interlacing sets. With a set interleaving the zeros of $p(\lambda)$ (and the eigenvalues of T_n) available, the subsequent approximations to these zeros (and to these eigenvalues) are obtained by using more customary techniques of [BOT]; for the sake of completeness, we will elaborate a modification of these techniques in this section and in section 4. (Some additional care is required here since we actually start not with an interlacing set but with its approximation.) The resulting algorithm consists of three main stages. At the first stage, specified in this section, we approximate some eigenvalues of T_n within a required error bound and cover each remaining eigenvalue by a real line interval containing no other eigenvalues of T_n. Such an eigenvalue may lie arbitrarily close to the end of the interval and, consequently, to other eigenvalues of T_n. However, the second stage ensures sufficiently strong isolation of all such eigenvalues from each other (by means of the bisection and the double exponential sieve algorithms). In the third stage, we rapidly approximate the isolated eigenvalues by means of Newton's iteration. The second and the third stages are described in section 4.

Now, let the set $\{d_0, \dots, d_n\}$ interleave the set Λ of the eigenvalues of T_n,

$$d_0 < \lambda_1 \le d_1 \le \lambda_2 \le d_2 \le \cdots \le d_{n-1} \le \lambda_n < d_n. \tag{3.1}$$

Let the pairs, d_i^-, d_i^+, of approximations to the real points d_i be given for $i = 1, \dots, n-1$, such that for a fixed Δ, we have

$$d_i^+ - d_i^- = 2\Delta, \quad d_i^- \le d_i \le d_i^+, \quad i = 0, \dots, n; \tag{3.2}$$

that is, the values d_i lie in the intervals

$$\mathcal{I}_i = \{\lambda : \ d_i^- \le \lambda \le d_i^+\}, \ i = 0, \dots, n. \tag{3.3}$$

Let us be given Δ, d_i^-, $d_i^+ = d_i^- + 2\Delta$ for $i = 0, \dots, n$, and the black box subroutine for the exact evaluation of the value at point λ of the characteristic polynomial of T_n,

$$p(\lambda) = \det(T_n - \lambda I). \tag{3.4}$$

In this section, for every eigenvalue λ_j of T_n (for $j = 1, \dots, n$), either we will compute its approximation, within the absolute error bound 2Δ, or we will determine that the interval

$$\mathcal{K}_j = \{\lambda : \ d_{j-1}^+ \le \lambda \le d_{j-}^-\} \tag{3.5}$$

contains λ_j and no other eigenvalues of T_n. This problem was solved in [BOT]; we propose an alternative solution.

With no loss of generality, we assume that

$$p(d_i^-)\ p(d_i^+) \neq 0, \quad i = 0, \dots, n, \tag{3.6}$$

and we will use the next definition and simple auxiliary results implied by (3.1)–(3.6).

DEFINITION 3.1. *The number of the eigenvalues of T_n in a real interval $\mathcal{I}$ is called the index of $\mathcal{I}$ and is denoted $c(\mathcal{I})$.*

PROPOSITION 3.1. *Any interval $\mathcal{K}_j$ cannot contain more than one eigenvalue of T_n.*

PROPOSITION 3.2. *$d_i^- \le \lambda_{i+1} \le d_{i+1}^+$, for $i = 0, 1, \dots, n-1$.*

COROLLARY 3.1. *If $\mathcal{I}_i \cap \mathcal{I}_{i+1} \cap \cdots \cap \mathcal{I}_{i+h} \neq \oslash$, then the points $\tilde{\lambda}_{i+j} = \frac{1}{2}(d_{i+j}^- + d_{i+j-1}^+)$ approximate λ_{i+j} within the absolute error bound 2Δ for $j = 1, 2, \dots, h$. Moreover, $h \le c(\mathcal{I})$, where $\mathcal{I} = \{\lambda :\ d_i^- \le \lambda \le d_{i+h}^+\}$. Furthermore, $c(\mathcal{I}) \le h + 2$, if in addition, $\mathcal{I}_{i-1} \cap \mathcal{I}_i = \mathcal{I}_{i+h} \cap \mathcal{I}_{i+h+1} = \oslash$.*

We will also use the following simple fact.

PROPOSITION 3.3. *Let $p(a)p(b) \neq 0$, $\mathcal{I} = \{\lambda :\ a \le \lambda \le b\}$. Then $p(a)p(b) < 0$ if and only if $c(\mathcal{I})$ is odd.*

ALGORITHM 3.1.

Input: Positive rational Δ, an integer n, and a set of rational numbers $D = \{d_i^-, d_i^+,\ \ i = 1, \dots, n-1\} \cup \{d_0^+ = -3(2^{-m}),\ \ d_n^- = 3(2^m)\}$ such that (3.1), (3.2), and (3.6) hold, and $d_0^+ \le \lambda_1,\ \ \lambda_n \le d_n^-$ (compare (2.1)).

Output: For every j, $j = 1, \dots, n$, either the interval $\mathcal{K}_j$ of (3.5) containing a unique eigenvalue λ_j of T_n or an approximation $\tilde{\lambda}_j$ to λ_j such that $|\lambda_j - \tilde{\lambda}_j| < 2\Delta$.

Stage 1 (form the union of the overlapping intervals): For $i = 0$ and for every i such that $d_i^+ < d_i^-$, determine the maximum $h = h(i)$ such that $d_{i+j-1}^+ \ge d_{i+j}^-$, for $j = 1, \dots, h$, letting $h(i) = 0$ if $d_i^+ < d_{i+1}^-$. Output $\tilde{\lambda}_{i+j} = (d_{i+j}^- + d_{i+j-1}^+)/2$, $j = 1, \dots, h$. (By virtue of Corollary 3.1, $|\lambda_{i+j} - \tilde{\lambda}_{i+j}| \le 2\Delta$.) Save the values $\eta_i = (d_i^- + d_i^+)/2$, $\nu_{i+h+1} = (d_{i+h}^- + d_{i+h}^+)/2$ as candidates for being approximations to λ_i and λ_{i+h+1}. Write

$$\mathcal{J}_{i,h} = \{\lambda :\ d_i^- \le \lambda \le d_{i+h}^+\} = \bigcup_{j=0}^{h} \mathcal{I}_{i+j} \tag{3.7}$$

and observe that

$$\lambda_i \le d_i^+ = d_i^- + 2\Delta,$$

$$\lambda_{i+h+1} \ge d_{i+h}^- \ge d_{i+h}^+ - 2\Delta,$$

so that

$$|\lambda_i - \eta_i| \le \Delta \ \text{ if } \ \lambda_i \in \mathcal{J}_{i,h},$$

$$|\lambda_{i+h+1} - \nu_{i+h+1}| \le \Delta \ \text{ if } \ \lambda_{i+h} \in \mathcal{I}_{i,h}.$$

Stage 2 (define the indices of the intervals $\mathcal{K}_j$ of (3.5) and $\mathcal{J}_{i,h}$ of (3.7)): Compute $p(d_i^-)$ and $p(d_i^+)$ for all i. Recall the relations (3.5)–(3.7), Propositions 3.1 and 3.3, and Corollary 3.1. For all i and j, define

$$c(\mathcal{K}_j) = \begin{cases} 0 & \text{if } p(d_{j+1}^-)\ p(d_j^+) > 0, \\ 1 & \text{otherwise,} \end{cases}$$

$$c(\mathcal{J}_{i,h}) = h+1 \ \text{ if } \begin{cases} \text{either } h \text{ is even and } p(d_i^-)p(d_{i+h}^+) < 0, \\ \text{or } h \text{ is odd and } p(d_i^-)p(d_{i+h}^+) > 0; \end{cases}$$

$$\text{otherwise, either } \ c(\mathcal{J}_{i,h}) - h = 0 \ \text{ or } \ c(\mathcal{J}_{i,h}) - h = 2. \tag{3.8}$$

Output the set of indices j such that $c(\mathcal{K}_j) = 1$, in which case $\lambda_{j-1} \in \mathcal{K}_j$.

Stage 3 (choose approximations among the candidate values): By the beginning of this stage, for every j, $j = 1, \dots, n$, it has been determined whether

$$\lambda_j \in \mathcal{K}_{j+1} \ (\text{see Stage 2})$$

or

$$|\lambda_j - \tilde{\lambda}_j| \le 2\Delta \ \ (\text{for } \tilde{\lambda}_j \ \text{ defined in Stage 1}),$$

or, otherwise, at least one of the next two bounds hold:

$$|\lambda_j - \eta_j| \le \Delta,$$

$$|\lambda_j - \nu_j| \le \Delta.$$

It remains to distinguish between the two latter cases and to choose an approximation to λ_j by one of the two candidates ν_j and η_j. This process relies on the two following simple rules:

(a) For every interval $\mathcal{J}_{i,h}$, of the two candidate values η_i and ν_{i+h+1}, exactly $c(\mathcal{J}_{i,h}) - h$ values should be selected as approximations within Δ to λ_i and/or λ_{i+h} (due to the observations made at the end of the description of Stage 1).

As soon as we decide about selecting one of the two candidate values η_i and ν_{i+h+1}, we apply the latter rule in order to decide if we should or should not select another of the two candidates (recall that we know the parity of $c(\mathcal{K}_j) - h$).

(b) For every j, $j = 1, \dots, n$, of the two consecutive candidate values ν_j and η_j, separated by the single interval $\mathcal{K}_j$, exactly $1 - c(\mathcal{K}_j)$ values should be selected as an approximation to the eigenvalue λ_j of T_n. (Indeed, $d_{j-1}^- < \nu_j < d_{j-1}^+ < d_j^- < \eta_j < d_j^+$ and $d_{j-1}^- \le \lambda_j \le d_j^+$, so that either $d_{j-1}^+ \le \lambda_j \le d_j^-$, and then $c(\mathcal{K}_j) = 1$, or $d_{j-1}^- \le \lambda_j \le d_{j-1}^+$, and then $|\nu_j - \lambda_j| \le \Delta$, or $d_j^- \le \lambda_j \le d_j^+$, and then $|\eta_j - \lambda_j| \le \Delta$.)

Rule (b) implies that neither of the values ν_j and η_j should be selected if $c(\mathcal{K}_j) = 1$; otherwise, we choose exactly one of them as an approximation to λ_j.

Recursive application of rules (a) and (b) completely defines the selection of the approximations to λ_j among all the candidate values η_j and ν_j for all j provided that we are given the values $p(d_i^+)$ and $p(d_i^-)$ for all i and that we know whether the leftmost interval $\mathcal{J}_{j,h}$ contains λ_j. The latter inclusion can be easily checked by using (3.1), (3.2), and Propositions 3.1–3.3. We shall decrease the parallel time of this

selection process by using the following equivalent procedure (the equivalence can be easily verified by inspection).

To select an approximation to λ_j by η_j or by ν_j, first partition the set S of all the intervals $\mathcal{J}_{i,h}$ satisfying (3.8) into the maximal subsets whose consecutive intervals are only interleaved with intervals $\mathcal{K}_j$ having indices 0 and with intervals $\mathcal{J}_{i,h}$ having indices $h+1$. Number the intervals in each maximal subset from left to right by $1, 2, \ldots$. For each subset, let $N(i,h)$ denote the number assigned to the interval $\mathcal{J}_{i,h}$ in this enumeration. Let $\delta = 0$ if there is no interval $\mathcal{K}_j$ having index 1 to the left of the subset and let $\delta = 1$ otherwise. Then determine the indices of the intervals of the subset as follows:

$$c(\mathcal{J}_{i,h}) = h + 1 + (-1)^{\delta + N(i,h)}. \tag{3.9}$$

Select both η_i and ν_{i+h+1} as approximations within Δ to λ_i and λ_{i+h+1} if $c(\mathcal{J}_{i,h}) = h+2$ and select none of them if $c(\mathcal{J}_{i,h}) = h$ (according to rule (a)).

Finally, for every i such that $c(\mathcal{J}_{i,h}) = h+1$, apply rule (b) to select one of the η_i and ν_{i+h+1} as an approximation within Δ to λ_i or to λ_{i+h+1}, respectively.

4. Approximation to the eigenvalues by using Newton's iterations, double exponential sieve, and the bisection method. In this section we will complement Algorithm 3.1 with an algorithm that approximates (within a fixed absolute output error Δ) an eigenvalue λ_j of T_n given a pair of real c and d, such that the interval $\mathcal{K} = \{\lambda : c \le \lambda \le d\}$ contains only this eigenvalue of T_n. We will apply the techniques of [BOT] and [R] based on the following result of [R].

THEOREM 4.1. *Let $p(x) = a_n \prod_{i=1}^{n}(x - \xi_i)$ be a polynomial. Let $x^{(0)} \in \mathbf{C}$ be such that $|x^{(0)} - \xi_1| \le |x^{(0)} - \xi_2| \le \cdots \le |x^{(0)} - \xi_n|$. If*

$$|x^{(0)} - \xi_1| < \frac{1}{5n^2}|x^{(0)} - \xi_2|, \tag{4.1}$$

then the sequence $x^{(i+1)} = x^{(i)} - p(x^{(i)})/p'(x^{(i)})$, $i = 0, 1, \ldots$, generated by Newton's method, converges to ξ_1; moreover, $|x^{(i)} - \xi_1| \le 2^{3-2^i}|x^{(0)} - \xi_1|$.

If we have an initial approximation $x^{(0)}$ such that $c < x^{(0)} < d$ and

$$|x^{(0)} - \lambda_j| \le \frac{1}{5n^2}\min\{x^{(0)} - c,\ d - x^{(0)}\}, \tag{4.2}$$

then we may invoke Theorem 4.1 and use Newton's iteration, thus arriving at the desired approximation to λ_j in $\lceil \log\log(0.8\frac{d-c}{\Delta n^2})\rceil$ Newton's steps. This observation leads us to the next procedure, which consists of the double exponential sieve process (Stages 2 and 3), bisection (Stage 4), together ensuring (4.1), and Newton's iteration (Stage 5), which outputs a real point $\tilde{\lambda}$ such that

$$|\tilde{\lambda} - \lambda| \le \Delta, \quad \lambda = \lambda_j. \tag{4.3}$$

ALGORITHM 4.1.

Input: Natural numbers $c < d$ such that $c < \lambda < d$ and a positive rational Δ.

Output: A rational $\tilde{\lambda}$ such that $|\lambda - \tilde{\lambda}| < \Delta$.

Stage 1. If $d - c < 2\Delta$, then output $\tilde{\lambda} = \frac{c+d}{2}$ and stop. Otherwise, set $\beta = 0$, compute $p(c)$ and $p((c+d)/2)$, set $c_0 = c$, $d_0 = d$, and restrict further computations to the interval $[c, \frac{c+d}{2}]$ if $p(c)p((c+d)/2) < 0$ and to the interval $[\frac{c+d}{2}, d]$ otherwise. Suppose, with no loss of generality, that $p(c) < 0$, $p((c+d)/2) > 0$, and perform the following stages.

Stage 2. Set $\gamma_0 = (c_0 + d_0)/2$ and apply the *bisection of the exponents* (also called the double exponential sieve) procedure to the interval $[c_0, (c_0 + d_0)/2]$; that is, successively evaluate $p(\gamma_i)$ for $\gamma_i = c_0 + (d_0 - c_0)2^{-2^i}$, $i = 1, 2, \ldots$, until either $\gamma_i - c_0 < 2\Delta$ or $\gamma_i - c_0 < \beta$ or $p(\gamma_i) \le 0$. In the first case, output $(c_0 + \gamma_i)/2$ and stop; in the second case, set $d_0 = \gamma_i$, $x_0 = (c_0 + \gamma_i)/2$ and go to Stage 4; otherwise, set $\beta = \gamma_i - c_0$ and go to the next stage. (Note that the second case may only occur for $\beta \ge 2\Delta > 0$, that is, not at the first pass through Stage 2.)

Stage 3. Set $c_1 = \gamma_i$ and $d_1 = \gamma_{i-1}$. (Note that $d_1 - c_1 \ge c_1 - c_0 \ge 2\Delta$.) Compute $p((c_1 + d_1)/2)$. If $p((c_1 + d_1)/2) < 0$, then set $c_0 = (c_1 + d_1)/2$, $d_0 = d_1$ and go to the next stage (in this case, $\lambda \in [c_0, d_0]$ and $d_0 - c_0 < \min\{c_0 - c,\ d - d_0\}$). Otherwise, set $c_0 = c_1$, $d_0 = (c_1 + d_1)/2$, and go to Stage 2.

Stage 4. Apply $\lceil \log(5n^2) \rceil$ bisection steps to $[c_0, d_0]$ in order to find a starting point $x^{(0)}$ satisfying (4.1).

Stage 5. Apply $\lceil \log\log(0.8\frac{d_0 - c_0}{\Delta n^2}) \rceil$ Newton's steps (starting with $x^{(0)}$) in order to arrive at an approximation $\tilde{\lambda}$ to λ such that $|\tilde{\lambda} - \lambda| < \Delta$.

Since β only grows, whereas $d_0 - c_0$ decreases by at least two times, in each recursive call to Stages 2 and 3, there can only be $O(\log^2(m + \log(\Delta^{-1})))$ such calls, and there can only be $O(\log(m + \log(\Delta^{-1})))$ Newton's iteration steps at Stage 5 provided that $-3(2^m) < c < d < 3(2^m)$.

5. Computing the values of the characteristic polynomial and its derivative at a set of points. We will concurrently apply Algorithm 4.1 to all the selected intervals, each containing a single eigenvalue of T_n. At any of the $O(\log^2(m + \log(\Delta^{(-1)})))$ steps we will have to compute the values $p(\lambda)$ and $p'(\lambda)$ at a set of at most n points. In this section we will devise an algorithm for computing the values of the characteristic polynomial $p(\lambda) = p_n(\lambda) = \det(T_n - \lambda I)$ of T_n and of its first derivative at a given set of n points.

Due to the tridiagonal structure of the matrix T_n, we have the following three-term recurrence for the polynomials $p_i(\lambda) = \det(T_i - \lambda I)$:

$$p_0(\lambda) = 1, \quad p_1(\lambda) = a_1 - \lambda,$$

$$p_{i+1}(\lambda) = (a_{i+1} - \lambda)\, p_i(\lambda) - b_i^2 p_{i-1}(\lambda), \quad i = n-1, n-2, \ldots, 1, \tag{5.1}$$

or in the equivalent matrix form,

$$\begin{pmatrix} p_{i+1}(\lambda) \\ p_i(\lambda) \end{pmatrix} = \begin{pmatrix} a_{i+1} - \lambda & -b_i^2 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} p_i(\lambda) \\ p_{i-1}(\lambda) \end{pmatrix},$$

so that

$$\begin{pmatrix} p_{i+1}(\lambda) \\ p_i(\lambda) \end{pmatrix} = F_i \ldots F_0 \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \tag{5.2}$$

$$F_j = \begin{pmatrix} a_{j+1} - \lambda & -b_j^2 \\ 1 & 0 \end{pmatrix}, \quad j = 0, \ldots, i, \quad b_0 = 0. \tag{5.3}$$

Now, we will compute the coefficients of $p(\lambda)$. At this stage, we do not have to restrict our computation to the real or complex case; we may allow the input entries of T_n from any field $\mathbf{F}$.

ALGORITHM 5.1.

Input: A positive integer $n = 2^h$ (compare Remark 2.1), and $a_1, \dots, a_n, b_1, \dots, b_{n-1}$ (the entries of T_n, being the elements of a field $\mathbf{F}$).

Output: $\alpha_0, \dots, \alpha_n \in \mathbf{F}$, such that $p(\lambda) = \det(T_n - \lambda I) = \sum_{i=0}^{n} \alpha_i \lambda^i$.

Computation: First set $H_j^{(0)} = F_j$, $j = 0, \dots, n-1$; then, for $i = 1, 2, \dots, \log n = h$, compute $H_j^{(i)} = H_{2j+1}^{(i-1)} H_{2j}^{(i-1)}$, $j = 0, \dots, 2^{-i}n - 1$, output the coefficients of the polynomial $p(\lambda) = (1 \;\; 0) H_0^{(h)} \binom{1}{0}$.

The ith level of the computation is essentially reduced to $8n/2^i$ multiplications of the entries of the matrices $H_j^{(i)}$, which are polynomials of degrees at most 2^i, and to four additions of the products. Over the complex or real fields $\mathbf{F}$, this means the $O_A(\log n, n)$ cost bound at each level i [AHU], [BM]; that is, the overall cost of Algorithm 5.1 is $O_A(\log^2 n, n)$.

Given the coefficients of $p(\lambda)$, and therefore, of $p'(\lambda)$, we may compute the values of both $p(\lambda)$ and $p'(\lambda)$ on a fixed set of n points, at the cost $O_A(\log^2 n \log\log n, n/\log\log n)$ (see [AHU], [BM], [RT]).

Now recall that we apply Newton's iteration to approximate the n eigenvalues of T_n. Each Newton's iteration step essentially amounts to computing the ratio $p(x)/p'(x)$ at n points, which now means the cost $O_A(\log^2 n \log\log n, n/\log\log n)$, and this is also an estimate for the cost of performing Algorithm 3.1.

Remark 5.1. Given the matrix T_n and a scalar x, $p(\lambda)$ can be evaluated at $\lambda = x$ at the cost $O_A(\log n, n)$. It is sufficient to apply Algorithm 5.1 replacing λ by $\lambda + x$ and performing the computation modulo λ. The same computation modulo λ^2 outputs the linear polynomial $p(x) + p'(x)\lambda$. Thus, given an $n \times n$ rst-matrix T_n, we may compute $p(\lambda)$ and $p'(\lambda)$ at $O(n)$ points at the cost $O_A(\log n, n^2)$, avoiding the evaluation of the coefficients of $p(\lambda)$. Even though the number of processors grows, the numerical stability of the computations is greatly improved in this way, which is the basis of the practical modification of our algorithm presented in [BP92]. The latter algorithm actually uses a slightly different modification of Algorithm 5.1, which yields the same output values of $p(\lambda)$ and $p'(\lambda)$ at the same computation cost but in, numerically, a more stable way. Besides improved numerical stability, we have also modified (in [BP92]) the isolation stage so as to bound the *relative* output errors, which is important for practical implementation.

6. The main algorithm. By using the tools and steps described in the preceding sections, we will now devise our main algorithm for the approximation of the eigenvalues of an $n \times n$ rst-matrix T_n with integer entries. This is a divide-and-conquer algorithm, which recursively reduces the original computational problem to two problems of half-size.

ALGORITHM 6.1.

Input: Two positive integers m and n, positive u; n integers $a_1, \dots, a_n$ and $n-1$ nonzero integers $b_1, \dots, b_{n-1}$, such that n is a power of 2 (compare Remark 2.1), $|a_i|$, $|b_i| \le 2^m$. (This input defines an rst-matrix T_n and the tolerance 2^{-u} to the output errors.)

Output: Reals $\gamma_1, \dots, \gamma_n$ such that $|\gamma_i - \lambda_i| < 2^{-u}$, where λ_i are the eigenvalues of T_n, $i = 1, \dots, n$.

Stage 1. Compute the coefficients of the characteristic polynomial of T_n by applying Algorithm 5.1.

Stage 2. Apply Algorithm 6.1 to the input set

$$u+1,\ m,\ \frac{n}{2},\ a_1,\dots,a_{\frac{n}{2}-1},a_{\frac{n}{2}}-b_{\frac{n}{2}},b_1,\dots,b_{\frac{n}{2}-1},$$

which defines an rst-matrix $S_{\frac{n}{2}}$, and to the input set

$$u+1,\ m,\ \frac{n}{2},\ a_{\frac{n}{2}+1}-b_{\frac{n}{2}},a_{\frac{n}{2}+2},\dots,a_n,b_{\frac{n}{2}},\dots,b_{n-1},$$

which defines an rst-matrix $R_{\frac{n}{2}}$, thus obtaining approximations $\delta_1 \le \delta_2 \le \cdots \le \delta_n$ to the eigenvalues of $S_{\frac{n}{2}}$ and $R_{\frac{n}{2}}$ within the absolute error bound $\Delta = 2^{-u-1}$. (The matrices $S_{\frac{n}{2}}$ and $R_{\frac{n}{2}}$ have been defined in Theorem 2.3.)

Stage 3. Recall that the set of all the eigenvalues of $S_{\frac{n}{2}}$ and $R_{\frac{n}{2}}$ interleaves the set $\{\lambda_i\}$ (see Theorem 2.3), set $d_i^+ = \delta_i + \Delta$, $d_i^- = \delta_i - \Delta$, and apply Algorithms 3.1 and 4.1, to obtain $\gamma_1,\dots,\gamma_n$ such that $|\gamma_i - \lambda_i| < 2^{-u}$.

The overall cost of performing the algorithm is $O_A(\log^3 n \log\log n(\log^2 b + \log n), n/\log\log n)$, $b = u+m$, but this bound can be decreased to $O_A(\log^2 n \log\log n(\log^2 b + \log n), n/\log\log n)$, since we need the output of Algorithm 4.1 with a precision lower than b bits until we arrive at the stage of approximating the eigenvalues of the original matix T_n (see the details in [BOT]).

Remark 6.1. An equivalent version of Algorithm 6.1 can be obtained by replacing the matrices $S_{\frac{n}{2}}$ and $R_{\frac{n}{2}}$ at Stage 3 by the matrices $T_{\frac{n}{2}}$ and $\hat{T}_{\frac{n}{2}-1}$ of part (b) of Theorem 2.2.

7. Bit-complexity estimates. We will prove the following result.

THEOREM 7.1. *Algorithm* 6.1 *can be implemented at the Boolean cost bounded by* $O_B(\log(nb)\log^2 n(\log^2 b + \log n)\log\log n \log\log(bn), n^2\frac{b+\log n}{\log\log n})$, *where* $|a_i|$, $|b_i| \le 2^m$, $b = m+u$.

The most expensive computations in Algorithm 6.1 are the evaluation of the coefficients of the characteristic polynomial $p(\lambda)$ of T_n at Stage 1, performed by means of Algorithm 5.1, and the evaluation at Stage 3 of the values of $p(\lambda)$ and $p'(\lambda)$ on a set of n points, performed by means of the multipoint polynomial evaluation algorithm of [AHU], [BM].

We will use the following auxiliary result.

THEOREM 7.2. *Let* n *be a power of* 2, k *a positive integer,* $p = 2^{nk/2}+1$, $u(x)$ *and* $v(x)$ *two polynomials with coefficients in* $\mathbf{Z}_p$ *(*$\mathbf{Z}_p$ *is the ring of integers with arithmetic modulo* p*),* $\deg v(x) = n_2 \le \deg u(x) = n_1 = O(n)$. *Then the coefficients of the polynomial*

$$w(x) = u(x)v(x)$$

can be computed modulo p *at the Boolean cost* $O_B(\log n \log(nk)\log\log(nk), n^2k)$. *Moreover, if the polynomial* $v(x)$ *is monic, the coefficients of the two polynomials* $q(x)$ *and* $r(x)$ *such that*

$$u(x) = v(x)q(x) + r(x)$$

and degree$(r(x)) < n_2$ *can be computed modulo* p *at the Boolean cost bounded by* $O_B(\log n \log\log n \log(nk)\log\log(nk), \frac{n^2k}{\log\log n})$.

Proof. We just need to combine the known estimates from [AHU] and [RT]. The Boolean cost of each arithmetic operation in $\mathbf{Z}_p$ is $O_B(\log d \log\log d, d)$, $d = \lceil \log p \rceil$

(see [AHU, p. 226], [RT]). Moreover, $2^{jk} \neq 1 \bmod p$, $j = 1, \ldots, n-1$, and $2^{nk} = 1 \bmod p$; that is, 2^k is a primitive nth root of 1 in $\mathbf{Z}_p$, and any integer power of 2, in particular, n, has its inverse in $\mathbf{Z}_p$. Therefore, in $\mathbf{Z}_p$, we may compute the product of a pair of nth-degree polynomials, by means of three FFTs, at the arithmetic cost $O_A(\log n, n)$ and, therefore, at the Boolean cost $O_B(\log n \log(nk) \log\log(nk), n^2 k)$. Finally, no divisions are needed for computing $q(x)$ and $r(x)$, since $v(x)$ is monic, and the computation can be performed in $\mathbf{Z}_p$ by using the polynomial division algorithm of [RT]. We arrive at the cost bounds $O_A(\log n \log\log n, n/\log\log n)$ and, therefore,

$$O_B(\log n \log(nk) \log\log n \log\log(nk), n^2 k/\log\log n). \qquad \square$$

Now, recall that the ith stage of Algorithm 5.1 (which evaluates the coefficients of the characteristic polynomial of T_n) consists in evaluating $8(n/2^i)$ products of pairs of polynomials of degrees at most 2^{i-1} and having integer coefficients whose absolute values are less than $2^{i+m2^{i+1}}$, for $i = 1, 2, \ldots, \log n$. Such coefficients are well defined by their values modulo $p = 2^{nk/2} + 1$, already for $k = 4m + 4$. Therefore, we may use integer arithmetic modulo p, apply Theorem 7.2, and deduce that the cost of performing Algorithm 5.1 is given by

$$O_B(\log^2 n \log(nm) \log\log(nm), n^2 m).$$

This bound is dominated by the cost bound of Theorem 7.1, since $b = m + u$, $u > 0$, and Brent's principle enables us to multiply the time bound by $s = \log\log n$ and simultaneously divide the processor bound by s.

Now, consider the evaluation of $p(x)$ and $p'(x)$ at a set of points $\gamma_1, \gamma_2, \ldots, \gamma_n$. We first observe that $|\gamma_i| \leq 2^{m+2}$, since γ_i, for $i = 1, \ldots, n$, are approximations, within absolute errors at most $2^{-u} \leq 1$, to the eigenvalues of rst-matrices having the 1-norms at most $3(2^m)$. Moreover, we may consider γ_i a binary value, $\gamma_i = y_i/2^{u+\log n}$, where y_i is an integer, $|y_i| \leq 2^{m+2+u+\log n}$, since we are looking for an approximation within the absolute error bound $2^{-u-\log n}$. Therefore, the evaluation of $p(\gamma_i)$ and $p'(\gamma_i)$ can be kept within the set of integers in the following way.

Consider $Q(y) = \sum_{i=-1}^{n} \alpha_i 2^{(u+\log n)(n-i)} y^i$, where $p(x) = \sum_{i=0}^{n} \alpha_i x^i$. The polynomial $Q(y)$ has integer coefficients, each represented with at most $n(m+u+2\log n)$ bits, and satisfies the following relation: $Q(y) = 2^{(u+\log n)n} p(x)$, $y = 2^{u+\log n} x$. Moreover, the size of the integers $Q(y_i)$ is bounded as follows:

$$|Q(y_i)| \leq 2^{n(2m+2u+3\log n+2)} = 2^{nk/2}, \quad k = O(m+u+\log n). \tag{7.1}$$

Now, we apply the known algorithm for multipoint polynomial evaluation [AHU], [BM], where we use integer arithmetic modulo $p = 2^{nk/2} + 1$ in the following way.

Recursively compute the coefficients of the monic polynomials $S_i^{(j)}$ of degrees 2^j,

$$S_k^{(0)} = y - y_k, \quad k = 1, \ldots, n,$$

$$S_i^{(j)} = S_{2i-1}^{(j-1)} S_{2i}^{(j-1)}, \quad i = 1, \ldots, \frac{n}{2^j}, \quad j = 1, \ldots, \log n - 1.$$

Recursively compute the coefficients of the polynomials $r_k^{(j)}$ of degrees at most $n/2^j$,

$$r_1^{(0)} = Q(y),$$

$$r_{2i-1}^{(j)} = r_i^{(j-l)} \bmod S_{2i-1}^{(\log n - j)},$$

$$r_{2i}^{(j)} = r_i^{(j-l)} \bmod S_{2i}^{(\log n - j)},$$

$i = 1, \dots, 2^{j-1}$, $j = 1, \dots, \log n$, such that $r_k^{(\log n)} = Q(y_k)$, $k = 1, \dots, n$. Combining the relations (7.1) and Theorem 7.1 and recalling that $b = m + \mu$, we deduce that such a multipoint polynomial evaluation can be performed at the overall cost

$$O_A(\log^2 n \log\log n, n/\log\log n)$$

and, consequently,

$$O_B(\log^2 n \log\log n \log(nb) \log\log(nb), n^2 b/\log\log n).$$

Therefore, Algorithm 6.1 can be carried out at the cost

$$O_B(\log^2 n \log(nb)(\log^2 b + \log n) \log\log n \log\log(nb), n^2 b/\log\log n),$$

performing a total of

$$O(n^2 b \log^2 n \log(nb)(\log^2 b + \log n) \log\log(nb))$$

bit-operations.

8. Discussion. We keep our results and proofs in their original form (cf. also [BP91]) assuming the PRAM models, though this assumption is not pertinent to the efficiency of our algorithms. Since the time of the submission of the present paper, some related results have appeared. The recent divide-and-conquer algorithms of [P95], [P96] approximate within 2^{-b} all the n complex zeros of any nth-degree polynomial, with its zeros in the unit disc, by using $O(n \log^2 b \log^2 n)$ arithmetic operations or $O((b+n)n^2 \log^2 n \log(bn) \log\log(bn))$ bit-operations; moreover, these algorithms have NC- and processor-efficient parallelization. The latter complexity bounds apply to any polynomial and are only slightly inferior to the current record bounds of the [BOT] and the present paper, which are restricted to the case where all the zeros of $p(x)$ are real. Combining the results of [P95], [P96] with the known algorithms for computing the coefficients of the characteristic polynomial of a general $n \times n$ matrix A gives an NC- and processor-efficient algorithm for the unsymmetric eigenvalue problem for A. On the other hand, the algorithms of [P95], [P96] do not extend directly to approximating the eigenvalues of unsymmetric matrices; extension of the divide-and-conquer techniques to the latter problem is a challenging open problem of practical importance.

Some minor improvements of the parallel complexity estimates of the present paper are possible, for instance, due to the recent improvement of parallel polynomial division achieved in [BP93] or via replacement of some integer divisions by multiplications.

The algorithm of this paper has several attractive features for its practical application; in particular, its computational cost is low, and its rapid convergence is guaranteed even where the input rst-matrix has clustered eigenvalues. The only major obstacle for the practical implementation is the stage of fast multipoint polynomial evaluation, which is known to be numerically unstable. There are two ways out of this difficulty, not counting recent progress in improving multipoint polynomial evaluation

(cf. [P95a], [PSLT], [PZHY]). One way, elaborated upon in [BP92], [B], and [BG1], proceeds by replacing the latter stage by slower but numerically stable computation. Another way, proposed and elaborated in [GE], relies on replacing the stage of multipoint polynomial evaluation by solving the associated secular equation by means of the multipole algorithm of [Ro85]. Unfortunately, [GE] deceptively claims its contribution to decreasing the known estimates for the computational complexity of the symmetric tridiagonal eigenproblem. In fact, the paper [GE] contains neither computational complexity estimates nor proper analysis of the case of clustered eigenvalues (effectively treated by the techniques of [BOT], [BP91], and [BP92]). Furthermore, [GE] fails to inform its readers about the existence of the *much earlier* papers [BOT], [BP91], and [BP92], with faster algorithms for the rst-eigenvalues, whereas comparison and, perhaps, combination of the techniques and the results of these papers with ones of [GE] could be informative and useful for the study of the symmetric tridiagonal eigenproblem. We also recall the papers [R93] (as the rediscovery of [BOT]) and [R97], whose main result (on multipoint polynomial evaluation) repeats one of [PSLT] and [PZHY].

Appendix A.

A.1. Reduction of approximating polynomial zeros to approximating matrix eigenvalues. Let $p(x)$ be a polynomial of a degree n having integer coefficients in the range from 2^m to 2^m and having only real zeros. There exist many $n \times n$ rst-matrices T_n whose characteristic polynomials are proportional to $p(x)$, that is, equal to $p(x) = p_n(x)$ after their appropriate normalization. We may specify T_n much better if, in addition to $p_n(x)$, we will fix the characteristic polynomials $p_{n-1}(x)$ of T_{n-1}, the $(n-1) \times (n-1)$ leading principal submatrix of T_n, and apply the extended Euclidean algorithm to $p_n(x)$ and $p_{n-1}(x)$. Suppose that we have chosen $p_{n-1}(x)$ such that this algorithm performs all its $n-1$ recursive steps, producing the $(n-1-i)$th-degree polynomial in the ith step, for $i = 1, \dots, n-1$. (This holds, in particular, if $p(x)$ has only real zeros and if $p_{n-1}(x) = -p'(x)$.) Then, due to (5.1), $p(x) = p_n(x)$ is the characteristic polynomial of an rst-matrix T_n whose entries a_i, b_i satisfy (5.1) for all i, and such a matrix T_n is defined uniquely, except that we may vary the signs of b_i as we like.

Let us analyze the complexity and errors of these computations assuming that $p_{n-1}(x) = -p'(x)$. Then the computation by the extended Euclidean algorithm can be performed at the cost bounded by $O_A(n \log^2 n, 1)$ [AHU] or $O_A(\log^3 n, n^2/\log n)$ [BP94]. Combining these bounds with the cost bounds of Algorithm 6.1 implies the estimates $O_A(n \log^2 n(\log^2 b + \log n), 1)$ and $O_A(\log^2 n(\log^2 b + \log n), (n/\log b)^2/\log n)$ for the sequential and parallel arithmetic complexity of approximating the zeros of a polynomial having only real zeros (compare [BOT]), and similarly, the sequential and parallel Boolean complexity estimates for approximating the zeros of $p(x)$ can be reduced to the estimates for Boolean complexity of the extended Euclidean computations and approximating the eigenvalues of T_n. The estimates for the sequential Boolean complexity and for the parallel Boolean time of the latter stage dominate the respective estimates for the overall complexity of approximating the zeros of $p(x)$, whereas the opposite is true for the bit-serial processor bound.

We will conclude this part of the appendix with two theorems that will enable us to bound the precision of the entries of T_n remaining within a prescribed tolerance to the errors of approximating the eigenvalues of T_n. As before, let a_i, $i = 1, \dots, n$, and b_j, $j = 1, \dots, n-1$, denote the diagonal and the subdiagonal entries of T_n, respectively. We do not assume any bounds on a_i and b_j but will deduce them.

THEOREM A.1. *The entries of T_n have absolute values at most $2^{m+1.5}$.*

Proof. Due to the Cauchy well-known bounds on the magnitudes of polynomial zeros, the zeros of $p(x)$ have absolute values at most $1+2^m < 2^{m+1}$. Due to Theorem 2.1, for $r = 1$, the diagonal entries of T_n, that is, $a_1, \dots, a_n$, have absolute values at most 2^{m+1}. By applying Theorem 2.1 with $r = 2$, we deduce that the eigenvalues of all the 2×2 submatrices

$$\begin{pmatrix} a_i & b_i \\ b_i & a_{i+1} \end{pmatrix}$$

have absolute values at most 2^{m+1}, that is, $|a_i a_{i+1} - b_i^2| \le 2^{2m+2}$; whence $|b_i| \le 2^{m+1.5}$. □

On the other hand, the eigenvalues of T_n are not very sensitive to the pertubation of the entries.

THEOREM A.2. *Let $\tilde{T}_n$ be an rst-matrix having diagonal and subdiagonal entries $\tilde{a}_i$, $i = 1, \dots, n$, and $\tilde{b}_j$, $j = 1, \dots, n-1$, respectively, such that $|\tilde{a}_i - a_i|$, $|\tilde{b}_j - b_j| \le 2^{-\nu}$, $i = 1, \dots, n$; $j = 1, \dots, n-1$. Then for any eigenvalue λ_i of T_n, there exists an eigenvalue $\tilde{\lambda}_i$ of $\tilde{T}_n$ such that $|\tilde{\lambda}_i - \lambda_i| \le 3(2^{-\nu})$.*

Proof. Theorem A.2 follows from the Bauer–Fike theorem (see [GL, p. 342]) applied for the Euclidean norm, since $\|\tilde{T} - T\|_2 \le 3(2^{-\nu})$. □

From the above theorems, we deduce that the rst-matrix $\tilde{T}_n$ obtained by setting $\tilde{a}_i = \lceil 2^\nu a_i \rceil$, $\tilde{b}_j = \lceil 2^\nu b_j \rceil$ has integer entries with absolute values at most $2^{\nu+m+1.5}$; furthermore, its eigenvalues, divided by $2^{-\nu}$, yield approximations $\tilde{\lambda}_i$ to the eigenvalues of T_n such that $|\tilde{\lambda}_i - \lambda_i| \le 3(2^{-\nu})$. Thus, to insure the latter bound, it suffices to compute the entries of T_n with the precision of $\lceil \nu + m + 1.5 \rceil$ bits.

A.2. Reduction of a Hermitian or real symmetric matrix to the tridiagonal form. Various randomized techniques are well known [GL] for the reduction of an $n \times n$ Hermitian or real symmetric matrix A to an rst-matrix T_n via similarity transformations (which leave invariant the eigenvalues and the characteristic polynomial of A). In [P87] and [BP94, Proposition 5.4, p. 325], a parallel implementation of such a tridiagonal reduction is shown and is analyzed. In particular, in the implementation of [BP94], tridiagonal reduction is essentially reduced

(a) to computation of the $2n + 1$ scalars $h_i = \vec{p}^T A^i \vec{q}$, $i = 0, 1, \dots, 2n$, for two random column vectors $\vec{p}$ and $\vec{q}$, and

(b) to the LDL^T (triangular) factorization of the associated Hankel matrix $H = (h_{i,j})$, $h_{i,j} = h_{i+j}$, $i, j = 0, 1, \dots, n-1$.

The overall computational cost of such a reduction can be bounded by $O_A(\log^2 n, n^3/\log n)$ or, alternatively, by $O_A(\log^3 n, P(n)/\log n)$, provided that a pair of $n \times n$ matrices can be multiplied at the cost bounded by $O_A(\log n, P(n)/\log n)$, $P(n) = O(n^{2.38})$.

Acknowledgments. The authors thank Prasoon Tiwari for kindly supplying a copy of [BOT] and the referee for helpful comments.

REFERENCES

[AHU] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The design and analysis of computer algorithms*, Addison-Wesley, Reading, MA, 1976.

[B] D. BINI, *Divide and conquer techniques for the polynomial root-finding problem*, in Proc. 1st World Congress of Nonlinear Analysts, Tampa, FL, 1992, V. Lakshmikantham, ed., Walter de Gruyter, Berlin, 1996, pp. 3885–3896.

[BFKT] M. Ben-Or, E. Feig, D. Kozen, and P. Tiwari, *A fast parallel algorithm for determining all roots of a polynomial with real roots*, SIAM J. Comput., 17 (1988), pp. 1081–1092.

[BG1] D. Bini and L. Gemigniani, *Iteration schemes for the divide-and-conquer eigenvalue solver*, Numer. Math., 67 (1994), pp. 403–425.

[BM] A. Borodin and I. Munro, *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier, New York, 1975.

[BNS] J. R. Bunch, C. P. Nielsen, and D. C. Sorensen, *Rank-one modification of the symmetric eigenproblem*, Numer. Math., 31 (1978), pp. 31–48.

[BOT] M. Ben-Or and P. Tiwari, *Simple algorithm for approximating all roots of a polynomial with real roots*, J. Complexity, 6 (1990), pp. 417–442.

[BP91] D. Bini and V. Y. Pan, *Parallel complexity of tridiagonal symmetric eigenvalue problem*, in Proc. 2nd Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, and SIAM, Philadelphia, 1991, pp. 384–393.

[BP92] D. Bini and V. Y. Pan, *Practical improvement of the divide-and-conquer eigenvalue algorithms*, Computing, 48 (1992), pp. 109–123.

[BP93] D. Bini and V. Y. Pan, *Improved parallel polynomial division*, SIAM J. Comput., 22 (1993), pp. 617–627.

[BP94] D. Bini and V. Y. Pan, *Matrix and Polynomial Computations, Volume 1: Fundamental Algorithms*, Birkhauser, Boston, 1994.

[Br] R. P. Brent, *The parallel evaluation of general arithmetic expressions*, J. Assoc. Comput. Mach., 21 (1974), pp. 201–208.

[Cu] J. J. M. Cuppen, *A divide and conquer method for the symmetric tridiagonal eigenproblem*, Numer. Math., 36 (1981), pp. 177–195.

[GE] M. Gu and S. C. Eisenstat, *A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 172–191.

[GL] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins Univ. Press, Baltimore, MD, 1989.

[Pan 87] V. Y. Pan, *Complexity of parallel matrix computations*, Theoret. Comput. Sci., 54 (1987), pp. 65–85.

[P87] V. Y. Pan, *Sequential and parallel complexity of approximate evaluation of polynomial zeros*, Comput. Math. Appl., 14 (1987), pp. 591–622.

[P95] V. Y. Pan, *Optimal (up to polylog factors) sequential and parallel algorithms for approximating complex polynomial zeros*, in Proc. 27th Annual ACM Symposium on Theory of Computing, ACM, New York, 1995, pp. 741–750.

[P95a] V. Y. Pan, *An algebraic approach on approximate evaluation of a polynomial on a set of real points*, Adv. Comput. Math., 3 (1995), pp. 41–58.

[P96] V. Y. Pan, *Optimal and nearly optimal algorithm for approximating complex polynomial zeros*, Comput. Math. Appl., 31 (1996), pp. 97–138.

[Par] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[PSLT] V. Y. Pan, A. Sadikou, E. Landowne, and O. Tiga, *A new approach to fast polynomial interpolation and multipoint evaluation*, Comput. Math. Appl., 25 (1993), pp. 25–30.

[PZHY] V. Y. Pan, A. Zheng, X. Huang, and Y. Yu, *Fast multipoint polynomial evaluation and interpolation via computations with structured matrices*, Ann. Numer. Math., 4 (1997), pp. 483–510.

[Ro85] V. Rokhlin, *Rapid solution of integral equations of classical potential theory*, J. Comput. Phys., 60 (1985), pp. 187–207.

[RT] J. H. Reif and S. H. Tate, *Optimal size integer division circuits*, in Proc. 21st ACM Symposium on Theory of Computing, ACM Press, New York, 1989, pp. 264–270.

[R] J. Renegar, *On the worst-case arithmetic complexity of approximating zeros of polynomials*, J. Complexity, 3 (1987). pp. 90–113.

[R93] J. H. Reif, *An $O(n \log^3 n)$ algorithm for the real root problem*, in Proc. 34th Annual IEEE Symposium on Foundations on Computer Science, IEEE, Piscataway, NJ, 1993, pp. 626–635.

[R97] J. H. Reif, *Approximate complex polynomial evaluation in near constant work per point*, in Proc. 29th Annual ACM Symposium on Theory of Computing, ACM, New York, 1997, pp. 30–39.

SIAM J. COMPUT.
Vol. 27, No. 4, pp. 1116–1141, August 1998
© 1998 Society for Industrial and Applied Mathematics
011

COMPUTATIONAL COMPLEXITY AND KNOWLEDGE COMPLEXITY*

ODED GOLDREICH†, RAFAIL OSTROVSKY‡, AND EREZ PETRANK§

Abstract. We study the computational complexity of languages which have interactive proofs of logarithmic knowledge complexity. We show that all such languages can be recognized in $\mathcal{BPP}^{\mathcal{NP}}$. Prior to this work, for languages with greater-than-zero knowledge complexity only trivial computational complexity bounds were known. In the course of our proof, we relate statistical knowledge complexity to perfect knowledge complexity; specifically, we show that, for the honest verifier, these hierarchies coincide up to a logarithmic additive term.

Key words. zero knowledge, interactive proofs, knowledge complexity, randomness, complexity classes, cryptography.

AMS subject classification. 68Q15

PII. S0097539795280524

1. Introduction.

The notion of knowledge complexity was introduced in the seminal paper of Goldwasser, Micali, and Rackoff [GMR-85, GMR-89]. Knowledge complexity is intended to measure the *computational advantage* gained by interaction. A formulation of knowledge complexity, for the case that it is not zero, has appeared in [GP-91]. A very appealing suggestion, made by Goldwasser, Micali, and Rackoff, is to characterize languages according to the knowledge complexity of their interactive proof systems [GMR-89].

The lowest level of the knowledge-complexity hierarchy is the class of languages having interactive proofs of knowledge complexity zero, better known as zero knowledge. Actually, there are three hierarchies extending the three standard definitions of zero knowledge: *perfect*, *statistical*, and *computational*. Assuming the existence of one-way functions, the third hierarchy collapses; that is, the zero level of the *computational* knowledge-complexity hierarchy contains all languages having interactive proof systems [GMW-86, IY-87, B+ 88], and thus contains all levels of the (computational) knowledge-complexity hierarchy. In this paper we will be only interested in the other two hierarchies. Previous works have provided information only concerning the zero level of these hierarchies (see, for example, Fortnow [F-89] and Aiello and Håstad [AH-87]). Our main result is an upper bound on the computational complexity of languages having logarithmic (statistical) knowledge complexity; namely, we show that such languages are contained in $\mathcal{BPP}^{\mathcal{NP}}$.

We consider the (statistical) knowledge-complexity hierarchy to be a very natural

*Received by the editors January 18, 1995; accepted for publication June 12, 1996; published electronically May 19, 1998. An extended abstract of this paper appeared in the 26*th ACM Symposium on Theory of Computing* (STOC 94), Montreal, Quebec, Canada, May 23–25, 1994.
http://www.siam.org/journals/sicomp/27-4/28052.html

†Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel (oded@wisdom.weizmann.ac.il). This research was supported by grant 92-00226 from the United States—Israel Binational Science Foundation, Jerusalem, Israel.

‡Bell Communications Research, 445 South Street, Morristown, NJ 07960-6438 (rafail@bellcore.com). Part of this work was done at University of California at Berkeley and International Computer Science Institute at Berkeley and supported by an NSF postdoctoral fellowship and ICSI.

§DIMACS Center, P.O. Box 1179, Piscataway, NJ 08855-1179 (erez@dimacs.rutgers.edu). This work was done while the author was a graduate student at the Technion, Haifa, Israel.

one. Its lowest level resides in the second level of the polynomial-time hierarchy (cf. [F-89, AH-87, B-85]), whereas as a whole it covers all of $\mathcal{PSPACE}$ (cf. [LFKN-90, Sh-90]). Another hierarchy with a similar feature, which also deserves investigation, is the hierarchy of languages classified by the number of rounds in their ("shortest") interactive proof system. Interestingly, the latter hierarchy has a multiplicative collapse (cf. [BM-88]), whereas no such result is known for the knowledge-complexity hierarchy.

1.1. Background on knowledge complexity. Loosely speaking, an interactive-proof system for a language L is a two-party protocol, by which a powerful *prover* can "convince" a probabilistic polynomial-time *verifier* of membership in L but will fail (with high probability) when trying to fool the verifier into "accepting" nonmembers [GMR-89]. An interactive proof is called *zero knowledge* if the interaction of any probabilistic polynomial-time machine with the predetermined prover, on common input $x \in L$, can be "simulated" by a probabilistic polynomial-time machine (called the *simulator*), given only x [GMR-89]. We say that a probabilistic machine M *simulates* an interactive proof if the output distribution of M is *statistically close* to the distribution of the real interaction between the prover and the verifier.

The formulation of zero knowledge presented above is known as *statistical* (almost-perfect) zero knowledge. Alternative formulations of zero knowledge are *computational* zero knowledge and *perfect* zero knowledge. In this paper we concentrate on statistical zero knowledge and the knowledge-complexity hierarchy that generalizes it.

Loosely speaking, the knowledge complexity of a protocol Π is the number of oracle bits that are needed to simulate the protocol efficiently. Namely, we say that a prover leaks (at most) $k(\cdot)$ bits of knowledge to verifier V if there is a probabilistic polynomial-time oracle machine ("simulator") M such that on any input $x \in L$, machine M makes at most $k(|x|)$ oracle queries, and the output distribution of $M(x)$ is statistically close to the distribution of the conversations in the interaction between the prover and V. For a formal definition and further discussion, see section 2.2.

The knowledge complexity of a language is the minimum knowledge complexity of an interactive proof system for that language. We consider the knowledge complexity of a language to be a very natural parameter. Furthermore, the question of how this parameter relates to the complexity of deciding the language is a very fundamental question.

1.2. Previous work. The complexity of recognizing zero-knowledge languages was first bounded by Fortnow [F-89]. He showed that any language that admits a zero-knowledge interactive proof is in the class co$\mathcal{AM}$. Subsequently, Aiello and Håstad [AH-87] showed that these languages are also in $\mathcal{AM}$.

Bellare and Petrank [BP-92] bounded the computational complexity of languages which have *short* interactive proofs with *low* knowledge complexity. Specifically, they showed that if a language L has a $g(n)$-round interactive proof which leaks at most $k(n)$ bits of knowledge and if $k(n) \cdot g(n) = O(\log n)$, then the language can be recognized in $\mathcal{BPP}^{\mathcal{NP}}$. This result does not apply to the general class of low knowledge-complexity languages, since these languages might not have interactive proofs which are both of small round complexity and low knowledge complexity.

1.3. This work. In this work we extend the result of [BP-92], showing that any language having an interactive proof with logarithmic knowledge complexity can be recognized in $\mathcal{BPP}^{\mathcal{NP}}$. We recall that $\mathcal{BPP}^{\mathcal{NP}}$ is contained in the third level of the polynomial-time hierarchy ($\mathcal{PH}$). It is believed that $\mathcal{PH}$ is a proper subset of

$\mathcal{PSPACE}$. Thus, assuming $\mathcal{PH} \subsetneq \mathcal{PSPACE}$, our result yields the first proof that there exist languages in $\mathcal{PSPACE}$ which cannot be proven by an interactive proof that yields $O(\log n)$ bits of knowledge. In other words, there exist languages which do have interactive proofs but only interactive proofs with super-logarithmic knowledge complexity. We stress that prior to our work there was no indication that would contradict the possibility that all languages in $\mathcal{PSPACE}$ have interactive proofs which yield only *one bit of knowledge*.

Our proof that languages of logarithmic knowledge complexity are in $\mathcal{BPP}^{\mathcal{NP}}$ consists of two parts. In the first part, we show that the $\mathcal{BPP}^{\mathcal{NP}}$ procedure described by Bellare and Petrank [BP-92] is applicable for recognizing languages that have interactive proofs of logarithmic *perfect* knowledge complexity. To this end, we use a more careful analysis than the one used in [BP-92]. In the second part of our proof, we transform interactive proofs of *statistical* knowledge complexity $k(n)$ into interactive proofs of *perfect* knowledge complexity $k(n) + O(\log n)$. This transformation refers only to knowledge complexity with respect to the honest verifier, but this suffices since the first part of our proof applies to the knowledge complexity with respect to the honest verifier. Yet, the transformation is interesting for its own sake, and a few words are in place.

The question of whether statistical zero knowledge equals perfect zero knowledge is one of the most fundamental open problems regarding zero knowledge. The question has been open also for the case of zero knowledge with respect to the honest verifier. Our transformation implies, as a special case, that any statistical zero-knowledge interactive proof can be modified into an interactive proof of perfect knowledge complexity bounded by a logarithmic function. Following the conference presentation of our work, Aiello, Bellare, and Venkatesan showed that statistical zero knowledge coincides with negligible *on the average* perfect knowledge complexity [ABV-95]. Their result is stronger in two respects; it refers to all verifiers, not only the honest verifier, and it bounds the perfect knowledge complexity by a negligible function rather than by a logarithmic one. On the other hand, their result is weaker as it refers to a much more liberal notion of (perfect) knowledge complexity, that is, "average" knowledge complexity rather than "worst-case" knowledge complexity (as considered here).

1.4. Organization. In section 2, we present the main definitions referred to in the rest of the paper. These include the definition of an interactive proof system as well as the definition of its knowledge complexity. Section 3 provides an overview to our proof that languages having (statistical) knowledge complexity bounded by a logarithmic function reside in $\mathcal{BPP}^{\mathcal{NP}}$. The first part of our proof (i.e., the case of perfect knowledge complexity) is presented in section 4. The second part of our proof (i.e., the transformation of statistical knowledge complexity to perfect knowledge complexity) is presented in section 5. Some concluding remarks appear in section 6.

In Appendix A, we discuss a flaw in Fortnow's paper [F-89]. We stress that the main result of [F-89] as well as its main techniques remain valid.

2. Preliminaries. Throughout this paper we use n to denote the length of the input x. A function $f : \mathsf{N} \to [0, 1]$ is called *negligible* if for every positive polynomial p and all sufficiently large n's $f(n) < \frac{1}{p(n)}$. We will say that two distribution ensembles indexed by strings are *statistically close* if the statistical distance between them is a negligible function of the length of the index.

2.1. Interactive proofs. Let us recall the concept of interactive proofs presented by [GMR-89]. For formal definitions and motivating discussions the reader is

referred to [GMR-89]. A protocol between a (computationally unbounded) *prover* P and a (probabilistic polynomial-time) *verifier* V constitutes an *interactive proof* for a language L if there exists a negligible function $\epsilon : \mathsf{N} \to [0,1]$ such that the following hold.

1. *Completeness.* If $x \in L$ then
$$\Pr[(P,V)(x) \text{ accepts}] \geq 1 - \epsilon(n).$$
2. *Soundness.* If $x \notin L$ then for any prover P^*
$$\Pr[(P^*,V)(x) \text{ accepts}] \leq \epsilon(n).$$

2.2. Knowledge complexity. Throughout the rest of the paper, we only refer to knowledge complexity *with respect to the honest verifier*, namely, the ability to simulate the honest verifier's view of its interaction with the prover. (In the stronger definition, one considers the ability to simulate the point of view of *any efficient verifier* while interacting with the prover.)

We let $(P,V)(x)$ denote the random variable that represents V's view of the interaction with P on common input x. The view contains the verifier's random tape as well as the sequence of messages exchanged between the parties.

We begin by briefly recalling the definitions of perfect and statistical zero knowledge. A protocol (P,V) is *perfect zero knowledge* (resp., *statistical zero knowledge*) over a language L if there is a probabilistic polynomial-time simulator M such that for every $x \in L$ the random variable $M(x)$ is distributed identically to $(P,V)(x)$ (resp., the statistical difference between $M(x)$ and $(P,V)(x)$ is a negligible function in $|x|$).

Next, we present the definitions of perfect (resp., statistical) knowledge complexity which we use in the sequel. These definitions extend the definition of perfect (resp., statistical) zero knowledge, in the sense that knowledge complexity zero is exactly zero knowledge. Actually, there are two alternative formulations of knowledge complexity, called the *oracle version* and the *fraction version*. These formulations coincide at the zero level and differ by at most an additive constant otherwise [GP-91]. For further intuition and motivation see [GP-91]. It will be convenient to use both definitions in this paper.

By the *oracle formulation*, the knowledge complexity of a protocol (P,V) is the number of oracle (bit) queries that are needed to simulate the protocol efficiently as described in the following definition.

DEFINITION 2.1 (knowledge complexity—oracle version). *Let* k: $\mathsf{N} \to \mathsf{N}$. *We say that an interactive proof* (P,V) *for a language* L *has* perfect (resp., statistical) knowledge complexity $k(n)$ in the oracle sense *if there exists a probabilistic polynomial-time oracle machine* M *and an oracle* A *such that*

1. *on input* $x \in L$, *machine* M *queries the oracle* A *for at most* $k(|x|)$ *bits;*
2. *for each* $x \in L$, *machine* M^A *produces an output with probability at least* $\frac{1}{2}$, *and given that* M^A *halts with an output,* $M^A(x)$ *is* identically distributed (resp., statistically close) *to* $(P,V)(x)$.

In the *fraction formulation*, the simulator is not given any explicit help. Instead, one measures the density of the largest subspace of the simulator's executions (i.e., coins) which is identical (resp., statistically close) to the (P,V) distribution.

DEFINITION 2.2 (knowledge complexity—fraction version). *Let* ρ: $\mathsf{N} \to (0,1]$. *We say that an interactive proof* (P,V) *for a language* L *has* perfect (resp., statistical) knowledge complexity $\log_2(1/\rho(n))$ in the fraction sense *if there exists a probabilistic polynomial-time machine* M *with the following* good subspace *property. For any* $x \in L$

there is a subset of M's possible random tapes, denoted S_x, such that the following hold.

1. *The set S_x contains at least a $\rho(|x|)$ fraction of the set of all possible coin tosses of $M(x)$.*
2. *Conditioned on the event that $M(x)$'s coins fall in S_x, the random variable $M(x)$ is* identically distributed (resp., statistically close) *to $(P,V)(x)$. Namely, for the perfect case this means that for every $\bar{c}$,*

$$\text{Prob}(M(x,\omega)=\bar{c}\,|\,\omega\in S_x) = \text{Prob}((P,V)(x)=\bar{c}),$$

where $M(x,\omega)$ denotes the output of the simulator M on input x and coin tosses sequence ω.

As mentioned above, these two measures are almost equal.

THEOREM (see [GP-91]). *The fraction measure and the oracle measure are equal up to an additive constant.*

Since none of our results is sensitive to a difference of an additive constant in the measure, we ignore this difference in the subsequent definition as well as in the statement of our results.

DEFINITION 2.3 (knowledge-complexity classes). $\mathcal{PKC}(k(\cdot))$ (resp., $\mathcal{SKC}(k(\cdot))$) *denotes the class of languages having interactive proofs of* perfect (resp., statistical) *knowledge complexity $k(\cdot)$.*

2.3. The simulation-based prover. An important ingredient in our proof is the notion of a simulation-based prover, introduced by Fortnow [F-89]. Consider a simulator M that outputs conversations of an interaction between a prover P and a verifier V. We define a new prover P^*, called *the simulation-based prover*, which selects its messages according to the conditional probabilities induced by the simulation. Namely, on a partial history h of a conversation, P^* outputs a message α with probability

$$\text{Prob}(P^*(h)=\alpha) \stackrel{\text{def}}{=} \text{Prob}(M_{|h|+1}=h\circ\alpha \;\big|\; M_{|h|}=h),$$

where M_t denotes the distribution induced by M on t-long prefixes of conversations. (Here, the length of a prefix means the number of messages in it.)

It is important to note that the behavior of P^* is *not* necessarily close to the behavior of the original prover P. Specifically, if the knowledge complexity is greater than zero and we consider the simulator guaranteed by the fraction definition, then P^* and P might have quite a different behavior. Our main objective will be to show that even in this case P^* still behaves in a manner from which we can benefit.

3. Overview. Using Definition 2.3, we state the main result of this paper as follows.

MAIN THEOREM. $\mathcal{SKC}(O(\log(\cdot))) \subseteq \mathcal{BPP}^{\mathcal{NP}}$.

We recall that all that was previously known regarding the $\mathcal{SKC}(\cdot)$ hierarchy is $\mathcal{SKC}(0) \subseteq \mathcal{AM} \cap \text{co}\mathcal{AM}$ and $\mathcal{BPP} \subseteq \mathcal{SKC}(k) \subseteq \mathcal{SKC}(k+1) \subseteq \mathcal{PSPACE}$ for every $k : \mathsf{N} \mapsto \mathsf{N}$.

The Main Theorem is proven in two stages:

1. $\mathcal{PKC}(O(\log(\cdot))) \subseteq \mathcal{BPP}^{\mathcal{NP}}$ (see Theorem 1);
2. $\mathcal{SKC}\,(k(\cdot)) \subseteq \mathcal{PKC}\,(k(\cdot) + O(\log(\cdot)))$ for every $k : \mathsf{N} \mapsto \mathsf{N}$ (see Theorem 2).

In the rest of this section we make several remarks regarding the above theorems and provide an overview to their proofs.

3.1. On the definitions underlying our results.

Remark 1. Usually, the definition of interactive proofs is robust in the sense that setting the error probability to be bounded away from $\frac{1}{2}$ does not change their expressive power, since the error probability can be reduced by repetitions. However, this standard procedure is NOT applicable when knowledge complexity is measured, since (even sequential) repetitions may increase the knowledge complexity. The question is, thus, what is the *right* definition. The definition used in section 2.1 is quite standard and natural; it is certainly less arbitrary then setting the error to be some favorite constant (e.g., $\frac{1}{3}$) or function (e.g., 2^{-n}). Yet, our techniques yield nontrivial results also in case one defines interactive proofs with nonnegligible error probability (e.g., constant error probability). For example, languages having interactive proofs with error probability $\frac{1}{4}$ and perfect knowledge complexity 1 are also in $\mathcal{BPP}^{\mathcal{NP}}$. For more details see Appendix B.

Remark 2. In the definition used in section 2.1 we have allowed two-sided error probability, rather than insisting on "perfect completeness" (as is sometimes done). This strengthens our Main Theorem but weakens the statistical-to-perfect transformation (i.e., Theorem 2), since a transformation for the case of one-sided error implies a transformation for the two-sided case,[1] whereas the converse is not clear.

Remark 3. The definitions of knowledge complexity in section 2.2 refer to simulations of the honest verifier. Analogous definitions of knowledge complexity refer to simulations of arbitrary polynomial-time verifiers (cf. [GP-91]). Let us denote the corresponding classes by $\mathcal{PKC}^*(\cdot)$ and $\mathcal{SKC}^*(\cdot)$. Clearly, $\mathcal{PKC}^*(k(\cdot)) \subseteq \mathcal{PKC}(k(\cdot))$ and $\mathcal{SKC}^*(k(\cdot)) \subseteq \mathcal{SKC}(k(\cdot))$ for every $k : \mathsf{N} \mapsto \mathsf{N}$. Thus, our Main Theorem is only strengthened by referring to the honest-verifier classes, whereas Theorem 2 is arguably weaker than an analogous statement referring to the arbitrary-verifier classes.

3.2. The perfect case—Overview. Our proof of Theorem 1 follows the procedure suggested in [BP-92], which in turn follows the approach of [F-89, BMO-90, Ost-91] while introducing a new "uniform generation" procedure which builds on ideas of [Si-83, St-83, GS-89, JVV-86].

Suppose that (P, V) is an interactive proof of perfect knowledge complexity $k(n) = O(\log n)$ for the languages L, and let M be the simulator guaranteed by the fraction formulation (i.e., Definition 2.2). We consider the conversations of the original verifier V with the simulation-based-prover P^* (see the definition in section 2.3). We show that the probability that the interaction (P^*, V) is accepting is negligible if $x \notin L$ and greater than a polynomial fraction if $x \in L$. Our proof differs from [BP-92] in the analysis of the case $x \in L$ (and thus we get a stronger result although we use the same procedure). This separation between the cases $x \notin L$ and $x \in L$ can be amplified by sequential repetitions of the protocol (P^*, V). So it remains to observe that we can sample the (P^*, V) interactions in probabilistic polynomial time having access to an NP-oracle. This observation originates from [BP-92] and is justified as follows. Clearly, V's part of the interaction can be produced in polynomial time. Also, by the uniform generation procedure of [BP-92] we can implement P^* by a probabilistic polynomial-time oracle machine that has access to an NP-oracle. Thus, it remains only to analyze the accepting probability of (P^*, V) on input x.

[1] Suppose one is given a transformation for the one-sided case. Then, given a two-sided interactive proof of some statistical knowledge complexity one could first transform it to a one-sided error proof system of the same knowledge complexity (cf. [GMS-87]). Applying the transformation for the one-sided case to the resulting one-sided error-proof system yields an (one-sided error) interactive proof with the desired knowledge complexity.

The case $x \notin L$ follows trivially from the soundness condition of V. The challenging case is when $x \in L$. If $k(n) = 0$ this case is easy since P^* behaves exactly as P and so the completeness condition guarantees that x will be accepted with very high probability. However, in case $k(n) > 0$ this argument is not valid and the simulator-based prover may behave very differently from the prescribed prover. Note that it is possible to define a prover P^{**} based on the behavior of the simulator on the "good subspace" and that P^{**} will indeed behave as P. However, it is not clear if P^{**} can be implemented in a relatively efficient manner (e.g., by a probabilistic polynomial-time machine that has access to an NP-oracle). Thus, we need to analyze the behavior of (P^*, V) on $x \in L$. For the sake of simplicity, we consider here only the special case in which $(P, V)(x)$ is always accepting (i.e., "perfect" completeness). Recall that the deviation of P^* from the behavior of P is due to the fact that behavior of the former is conditioned on the entire probability space of the simulator, whereas the latter is conditioned on the "good subspace." In each case the next prover move is determined by the set of all simulator coins which match the current history of the interaction. For P^* this is the set of all coin tosses which may produce this history, whereas for P this is the set of all good coin tosses (i.e., coins in the "good subspace") which produce this history. We first observe that the key parameter for the analysis of P^* is the ratio between the size of the residual probability space of the simulator and the size of the residual space of good coins. Actually, we consider the reciprocal of the above ratio. We observe that the *expected value* of the latter ratio may only increase as a function of the history length, where the expectation is taken over all possible histories of fixed length as produced by a (P^*, V) interaction. Finally, we observe that the expected value of the ratio for a full interaction is a lower bound on the probability that P^* makes V accept the input, whereas for the empty interaction the ratio equals 2^{-k}.

3.3. The transformation—Overview. Our proof of Theorem 2 refers to the oracle formulation of knowledge complexity (see Definition 2.1). Suppose we are given a simulator which produces output that is *statistically close* to the real prover–verifier interaction. We will change both the interactive proof and its simulation *so that they produce exactly the same distribution.* We will take advantage of the fact that the prover in the interactive proof and the oracle that "assists" the simulator are both infinitely powerful. Thus, the modification to the prover's program and the augmentation to the oracle need not be efficiently computable. We stress that the modification to the simulator itself will be efficiently computable. Also, we maintain the original verifier (of the interactive proof), and thus the resulting interactive proof is still sound. Furthermore, the resulting interaction will be statistically close to the original one (on any $x \in L$) and, therefore, the completeness property of the original interactive proof is maintained (although the error probability here may increase by a negligible amount).

The key question is how can we modify the two relevant distributions so that they become identical rather than statistically close. The easy case is when some conversation is more likely in the simulation (than in the original prover–verifier interaction). This case is handled by providing the oracle with a candidate conversation and having the oracle decide probabilistically whether we should output this conversation or not. Thus, we can use one additional oracle query in order to lower the probability of conversations produced by the original simulator. However, the challenging case is when some conversation is less probable in the simulation (than in the original interaction). Using the oracle to produce such conversations is too costly, in terms of

query complexity, unless we consider average-case query complexity (as in [ABV-95]). Thus, we need a different approach. Our approach is to modify the original prover so that it truncates conversations at a point where they become less probable in the simulation. This truncation is also probabilistic. A new simulator, with the help of an augmented oracle, will have to detect the truncation point and produce truncated conversations with the same probability as they are produced in interaction with the new prover. In order to specify the truncation point we need to get a t-ary value from the oracle, where t is the total number of bits in the interaction. This is implemented using $\log_2 t$ queries giving rise to the additive logarithmic factor in the result of the theorem.

4. The perfect case. In this section we prove that the Main Theorem holds for the special case of *perfect* knowledge complexity.

THEOREM 1. $\mathcal{PKC}(O(\log n)) \subseteq \mathcal{BPP}^{\mathcal{NP}}$.

As stated above, our proof follows the procedure suggested in [BP-92]. Suppose that (P, V) is an interactive proof of perfect knowledge complexity $k(n) = O(\log n)$ for the languages L, and let M be the simulator guaranteed by Definition 2.2. Let us denote by P^* the simulation-based prover (for M); see section 2.3. Then the following holds.

LEMMA 4.1 (see [BP-92]). *P^* can be implemented by a probabilistic polynomial-time oracle machine that has access to an NP-oracle.*

LEMMA 4.2 (analysis of the behavior of P^*).

1. *If $x \in L$ then the probability that (P^*, V) outputs an accepting conversation is at least $\frac{1}{2} \cdot 2^{-k(|x|)}$.*
2. *If $x \notin L$ then the probability that (P^*, V) outputs an accepting conversation is negligible (in $|x|$).*

Remark. In [BP-92], a weaker lemma is proven. Specifically, they show that the probability that (P^*, V) outputs an accepting conversation on $x \in L$ is related to $2^{-k(|x|)\cdot t(|x|)}$, where $t(\cdot)$ is the number of rounds in the protocol. We stress that our lemma does not refer to the number of rounds which may be polynomial in $|x|$, whereas the weaker form of [BP-92] is meaningful only for logarithmic number of rounds (i.e., $t(n) = O(\log n)$).

4.1. Proof of Theorem 1. Combining Lemma 4.1 with the fact that V is probabilistic polynomial time and using Lemma 4.2, we obtain a probabilistic polynomial-time oracle machine A that when given access to an NP-oracle satisfies, for some polynomial p, the following statements.

1. If $x \in L$ then $\text{Prob}(A^{NP}(x)=1) \geq \frac{1}{p(|x|)}$.
2. If $x \notin L$ then $\text{Prob}(A^{NP}(x)=1) \leq \frac{1}{2p(|x|)}$.

(For example, $p(n) = 2^{k(n)+1}$ will do.) Using standard amplification, we conclude that $L \in \mathcal{BPP}^{\mathcal{NP}}$. □

4.2. Proof of Lemma 4.2. The second part of the lemma follows from the soundness property of V; namely, the probability that V accepts $x \notin L$ is negligible no matter what the prover does. We thus concentrate on the first part. We fix an arbitrary $x \in L$ for the rest of the proof and allow ourselves not to mention it in the sequel discussion and notation. Let $k = k(|x|)$ and q be the number of coin tosses made by M. We denote by $\Omega \stackrel{\text{def}}{=} \{0,1\}^q$ the set of all possible coin tosses, and by S the "good subspace" of M (i.e., S has density 2^{-k} in Ω and for ω chosen uniformly in S the simulator outputs exactly the distribution of the interaction (P, V)).

4.2.1. Motivation. Consider the conversations that are output by the simulator on coins $\omega \in S$. The probability of getting such a conversation when the simulator is run on ω uniformly selected in Ω is at least 2^{-k}. We *claim* that the probability to get these conversations in the interaction (P^*, V) is also at least 2^{-k}. This is not obvious, since the distribution produced by (P^*, V) may not be identical to the distribution produced by M on a uniformly selected $\omega \in \Omega$. Nor is it necessarily identical to the distribution produced by M on a uniformly selected $\omega \in S$. However, the prover's moves in (P^*, V) are distributed as in the case that the simulator selects ω uniformly in Ω, whereas the verifier's moves (in (P^*, V)) are distributed as in the case that the simulator selects ω uniformly in S. Thus, it should not be too surprising that the above claim can be proven.

However, we need more than the above claim. It is not enough that the (P^*, V) conversations have an origin in S, they must be *accepting* as well. (Note that this is not obvious since M simulates an interactive proof that may have two-sided error.) Again, the density of the accepting conversations in the "good subspace" of M is high, yet we need to show that this is the case also for the (P^*, V) interaction. Actually, we will show that the probability than a (P^*, V) conversation is accepting and "has an origin" in S is at least $\frac{1}{2} \cdot 2^{-k}$.

4.2.2. Preliminaries. Let us begin the formal argument with some notations. For each possible history of the interaction h, we define subsets of the random tapes of the simulator (i.e., subsets of Ω) as follows. Ω_h is the set of $\omega \in \Omega$ which causes the simulator to output a conversation with prefix h. S_h is the subset of ω's in Ω_h which are also in S. A_h is the set of ω's in S_h which are also accepting. Thus, letting $M_t(\omega)$ denote the t-message long prefix output by the simulator M on coins ω, we get

$$\begin{aligned}
\Omega_h &\stackrel{\text{def}}{=} \{\omega : M_{|h|}(\omega) = h\},\\
S_h &\stackrel{\text{def}}{=} \Omega_h \cap S,\\
A_h &\stackrel{\text{def}}{=} \{\omega \in S_h : M(\omega) \text{ is accepting}\}.
\end{aligned}$$

Let C be a random variable representing the (P^*, V) interaction, and χ be an indicator so that $\chi(\bar{c}) = 1$ if the conversation $\bar{c}$ is accepting and $\chi(\bar{c}) = 0$ otherwise. Our aim is to prove that $\text{Prob}(\chi(C) = 1) \geq \frac{1}{2} \cdot 2^{-k}$. Note that

$$\begin{aligned}
\text{Prob}(\chi(C) = 1) &= \sum_{\bar{c}} \text{Prob}(C = \bar{c}) \cdot \chi(\bar{c})\\
&\geq \sum_{\bar{c}} \text{Prob}(C = \bar{c}) \cdot \frac{|A_{\bar{c}}|}{|\Omega_{\bar{c}}|}.
\end{aligned}$$

The last expression is exactly the expectation value of $\frac{|A_c|}{|\Omega_c|}$. Thus, it suffices to show that

$$\text{Exp}_{\bar{c}} \left(\frac{|A_{\bar{c}}|}{|\Omega_{\bar{c}}|} \right) > \frac{1}{2} \cdot 2^{-k}, \tag{1}$$

where the expectation is over the possible conversations $\bar{c}$ as produced by the interaction (P^*, V). Once equation (1) is proven, we are done. Denote the empty history by λ. To prove equation (1) it suffices to prove that

$$\text{Exp}_{\bar{c}} \left(\frac{|A_{\bar{c}}|}{|\Omega_{\bar{c}}|} \cdot \frac{|A_{\bar{c}}|}{|S_{\bar{c}}|} \right) \geq \frac{|A_\lambda|}{|\Omega_\lambda|} \cdot \frac{|A_\lambda|}{|S_\lambda|} \tag{2}$$

since using equation (1), $\frac{|A_\lambda|}{|S_\lambda|} > \sqrt{\frac{1}{2}}$ and $\frac{|S_\lambda|}{|\Omega_\lambda|} \geq 2^{-k}$, we get

$$\begin{aligned}\mathrm{Exp}_{\bar{c}}\left(\frac{|A_{\bar{c}}|}{|\Omega_{\bar{c}}|}\right) &\geq \frac{|A_\lambda|}{|\Omega_\lambda|}\cdot\frac{|A_\lambda|}{|S_\lambda|}\\ &= \left(\frac{|A_\lambda|}{|S_\lambda|}\right)^2\cdot\frac{|S_\lambda|}{|\Omega_\lambda|}\\ &> \frac{1}{2}\cdot 2^{-k}.\end{aligned}$$

The proof of equation (2) is by induction on the number of rounds. Namely, for each round i we show that the expected value of $\frac{|A_h|}{|\Omega_h|}\cdot\frac{|A_h|}{|S_h|}$ over all possible histories h of i rounds (i.e., length i) is greater or equal to the expected value of this expression over all histories h' of $i-1$ rounds. In order to show the induction step we consider two cases:

1. the current step is by the prover (i.e., P^*); and
2. the current step is by the verifier (i.e., V).

In both cases we show, for any history h,

$$\mathrm{Exp}_m\left(\frac{|A_{h\circ m}|}{|\Omega_{h\circ m}|}\cdot\frac{|A_{h\circ m}|}{|S_{h\circ m}|}\right) \geq \frac{|A_h|}{|\Omega_h|}\cdot\frac{|A_h|}{|S_h|}, \tag{3}$$

where the expectation is over the possible current moves m, given history h, as produced by the interaction (P^*, V).

4.2.3. A technical claim. The following technical claim is used for deriving the inequalities in both cases.

CLAIM 4.3. *For $1 \leq i \leq n$, let x_i and y_i be positive reals. Then,*

$$\sum_{i=1}^{n}\frac{{x_i}^2}{y_i} \geq \frac{(\sum_{i=1}^{n} x_i)^2}{\sum_{i=1}^{n} y_i}.$$

Proof. The Cauchy–Schwarz inequality asserts

$$\left(\sum_{i=1}^{n}{a_i}^2\right)\cdot\left(\sum_{i=1}^{n}{b_i}^2\right) \geq \left(\sum_{i=1}^{n} a_i\cdot b_i\right)^2.$$

Setting $a_i \stackrel{\mathrm{def}}{=} \sqrt{y_i}$ (we can do this since y_i is positive) and $b_i \stackrel{\mathrm{def}}{=} \frac{x_i}{a_i}$ and rearranging the terms, we get the desired inequality. □

4.2.4. Prover step (denoted α). Using the fact that P^* is a simulation-based prover for M, we observe that given history h, the prover P^* sends α as its next message with probability exactly $\frac{|\Omega_{h\circ\alpha}|}{|\Omega_h|}$. Thus,

$$\begin{aligned}\mathrm{Exp}_\alpha\left(\frac{|A_{h\circ\alpha}|}{|\Omega_{h\circ\alpha}|}\cdot\frac{|A_{h\circ\alpha}|}{|S_{h\circ\alpha}|}\right) &= \sum_\alpha \frac{|\Omega_{h\circ\alpha}|}{|\Omega_h|}\cdot\frac{|A_{h\circ\alpha}|}{|\Omega_{h\circ\alpha}|}\cdot\frac{|A_{h\circ\alpha}|}{|S_{h\circ\alpha}|}\\ &= \frac{1}{|\Omega_h|}\cdot\sum_\alpha\frac{|A_{h\circ\alpha}|^2}{|S_{h\circ\alpha}|}\\ &\geq \frac{|A_h|}{|\Omega_h|}\cdot\frac{|A_h|}{|S_h|}.\end{aligned}$$

The inequality is justified by using Claim 4.3 and noting that $\sum_\alpha |A_{h\circ\alpha}| = |A_h|$ and $\sum_\alpha |S_{h\circ\alpha}| = |S_h|$.

4.2.5. Verifier step (denoted β). Using the perfectness of the simulation, when restricted to the good subspace S, we observe that given history h the verifier V sends β as its next message with probability exactly $\frac{|S_{h\circ\beta}|}{|S_h|}$. Thus,

$$\begin{aligned}\mathrm{Exp}_\beta\left(\frac{|A_{h\circ\beta}|}{|\Omega_{h\circ\beta}|}\cdot\frac{|A_{h\circ\beta}|}{|S_{h\circ\beta}|}\right)&=\sum_\beta\frac{|S_{h\circ\beta}|}{|S_h|}\cdot\frac{|A_{h\circ\beta}|}{|\Omega_{h\circ\beta}|}\cdot\frac{|A_{h\circ\beta}|}{|S_{h\circ\beta}|}\\&=\frac{1}{|S_h|}\cdot\sum_\beta\frac{|A_{h\circ\beta}|^2}{|\Omega_{h\circ\beta}|}\\&\geq\frac{|A_h|}{|\Omega_h|}\cdot\frac{|A_h|}{|S_h|}.\end{aligned}$$

The inequality is justified by using Claim 4.3 and noting that $\sum_\beta|A_{h\circ\beta}|=|A_h|$ and $\sum_\beta|\Omega_{h\circ\beta}|=|\Omega_h|$.

Having proven equation (3) for both cases, equation (2) follows and so does the lemma. □

5. The transformation. In this section we show how to transform *statistical* knowledge complexity into *perfect* knowledge complexity, incurring only a logarithmic additive term.

THEOREM 2. *For every (poly-time computable)* $k:\mathsf{N}\mapsto\mathsf{N}$,

$$\mathcal{SKC}\left(k(\cdot)\right)\subseteq\mathcal{PKC}\left(k(\cdot)+O(\log(\cdot))\right).$$

We stress again that these knowledge-complexity classes refer to the honest verifier and that we don't know whether such a result holds for the analogous knowledge-complexity classes referring to arbitrary (polynomial-time) verifiers.

The rest of this section is devoted to proving the above theorem. All the numbered claims appearing below are quite evident from the corresponding definitions and so the reader may skip their proofs (which are provided for sake of completeness). This holds also with respect to Claim 5.3.

5.1. Preliminaries. Let $L\in\mathcal{SKC}(k(\cdot))$, and let (P,V) be the guaranteed interactive proof. Without loss of generality, we may assume that all messages are of length 1. Here we use the oracle formulation of knowledge complexity (see Definition 2.1). Recall that Definition 2.1 only guarantees that the simulator produces output with probability at least $\frac{1}{2}$. Yet, employing Proposition 3.8 of [GP-91], we get that there exists an oracle machine M that, after asking $k(n)+2\log\log n$ queries, *always* produces an output so that the output is statistically close to the interaction of (P,V). Let A denote the associated oracle and let $M'\stackrel{\mathrm{def}}{=}M^A$. When we talk in the sequel of modifying M', what we actually mean is modifications to the code of M and augmentations of the oracle A. All the modifications in the code correspond to operations that can be performed in probabilistic polynomial time.

Let P' be the simulation-based prover induced by M'. Similarly, let V' be the simulator-based verifier induced by M'. A simulator-based verifier is defined analogously to the simulator-based prover. It is a fictitious entity which does not necessarily coincide with V. Thus, $M'(x)$ and $(P',V')(x)$ are identically distributed. In the rest of the presentation, we fix a generic input $x\in L$ and omit it from the notation.

Notation. Let $[A,B]_i$ be a random variable representing the i-message (i-bit) long prefix of the interaction between A and B (the common input x is implicit in

the notation). We denote by $A(h)$ the random variable representing the message sent by A after interaction history h. Thus, if the ith message is sent by A, we can write $[A,B]_{i-1} \circ A([A,B]_{i-1}) = [A,B]_i$. By $X \stackrel{s}{=} Y$ we denote the fact that the random variables X and Y are statistically close.

Using these notations we may write for every $h \in \{0,1\}^i$ and $\sigma \in \{0,1\}$,

$$\text{Prob}(P'(h) = \sigma) \;=\; \text{Prob}\left([M']_{i+1} = h \circ \sigma \mid [M']_i = h\right)$$

and, similarly,

$$\text{Prob}(V'(h) = \sigma) \;=\; \text{Prob}\left([M']_{i+1} = h \circ \sigma \mid [M']_i = h\right).$$

CLAIM 5.1 (analysis of the behavior of (P',V)). *The distribution induced by (P',V) is statistically close to the distributions induced by both $M' = (P',V')$ and (P,V).*

Proof. By definition, the distributions produced by $M' = (P',V')$ and (P,V) are statistically close. Thus, we have

$$[P,V]_i \stackrel{s}{=} [P',V']_i \quad \text{for every } i. \tag{4}$$

We prove that $[P',V]$ is statistically close to $[P',V']$ by induction on the length of the interaction. We stress that we will use the induction hypothesis only once in our proof of the induction step, and thus the statistical distance grows at most linearly with the number of induction steps. Since the number of induction steps is polynomial, the statistical distance at the last induction step (i.e., between the random variables representing the full interaction) is negligible. (Note that for every negligible function μ and any polynomial p the function $\mu'(n) \stackrel{\text{def}}{=} p(n) \cdot \mu(n)$ is negligible.)

We now prove the induction step. Assuming that $[P',V]_i \stackrel{s}{=} [P',V']_i$, we wish to prove it for $i+1$. We distinguish two cases. In case the $i+1$st move is by the prover, we get

$$\begin{aligned}
[P',V]_{i+1} &= [P',V]_i \circ P'([P',V]_i) \\
&\stackrel{s}{=} [P',V']_i \circ P'([P',V']_i) \\
&= [P',V']_{i+1},
\end{aligned}$$

where $\stackrel{s}{=}$ follows by the induction hypothesis. (Actually, we also use the fact that the statistical distance can only decrease when the same probabilistic process is applied to two random variables; specifically, the process here is $R(x) \stackrel{\text{def}}{=} x \circ P'(x)$.) In case the $i+1$st move is by the verifier, we get

$$\begin{aligned}
[P',V]_{i+1} &= [P',V]_i \circ V([P',V]_i) \\
&\stackrel{s}{=} [P',V']_i \circ V([P',V']_i) \\
&\stackrel{s}{=} [P,V]_i \circ V([P,V]_i) \\
&= [P,V]_{i+1} \\
&\stackrel{s}{=} [P',V']_{i+1},
\end{aligned}$$

where the first $\stackrel{s}{=}$ is justified by the induction hypothesis and the other two by equation (4). ∎

5.2. Motivating discussion. Note that the statistical difference between the interaction (P', V) and the simulation $M' = (P', V')$ is due solely to the difference between the proper verifier (i.e., V) and the verifier induced by the simulator (i.e., V'). This difference is due to V' putting too much probability weight on certain moves and thus also too little weight on their sibling messages (recall that a message in the interaction consists of a single bit). In what follows we deal with two cases.

The first case is when this difference between the behavior of V' (induced by M') and the behavior of the verifier V is "more than tiny." This case receives most of our attention. We are going to use the oracle in order to move weight from a verifier message β that gets too much weight (after a history h) to its sibling message $\beta \oplus 1$ that gets too little weight in the simulation. Specifically, when the new simulator M'' invokes M' and comes up with a conversation that has $h \circ \beta$ as a prefix, the simulator M'' (with the help of the oracle) will output a conversation with the prefix $h \circ (\beta \oplus 1)$ instead of outputting the original conversation. The simulator M'' will do this with probability that exactly compensates for the difference between V' and V. This leaves one problem. How does the new simulator M'' come up with a conversation that has a prefix $h \circ (\beta \oplus 1)$? The cost of letting the oracle supply the rest of the conversation (after the known prefix $h \circ (\beta \oplus 1)$) is too high. We adopt a "brutal" solution in which we truncate all conversations that have $h \circ (\beta \oplus 1)$ as a prefix. The truncation takes place both in the interaction (P'', V), where P'' stops the conversation after $\beta \oplus 1$ (with a special STOP message) and in the simulation where the oracle recognizes cases in which the simulator M'' should output a truncated conversation. These changes make M'' and V behave exactly the same on messages for which the difference between V' and V is more than tiny. Naturally, V immediately rejects when P'' stops the interaction abruptly, so we have to make sure that this change does not foil the ability of P'' to convince V on an input $x \in L$. It turns out that these truncations happen with negligible probability since such truncation is needed only when the difference between V and V' is more than tiny. Thus, P'' convinces V on $x \in L$ almost with the same probability as P' does.

The second possible case is that the difference between the behavior of V and V' is tiny. In this case, looking at a full conversation $\bar{c}$, we get that the tiny differences sum up to a small difference between the probability of $\bar{c}$ in the distributions of M' and of (P', V). We correct these differences by lowering the probabilities of all conversations in the new simulator. The probability of each conversation is lowered so that its relative weight (relative to all other conversations) is equal to its relative weight in the interaction (P'', V). Technically, this is done by M'' not producing an output in certain cases that M' did produce an output.

Technical remark. The oracle can be used to allow the simulator to toss bias coins even when the simulator does not "know" the bias. Suppose that the simulator needs to toss a coin so that it comes up *head* with probability $\frac{N}{2^m}$, where $N < 2^m$ and both N and m are integers. The simulator supplies the oracle with a uniformly chosen $r \in \{0,1\}^m$ and the oracle answers *head* if r is among the first N strings in $\{0,1\}^m$ and *tail* otherwise. A similar procedure is applicable for implementing a lottery with more than two a priori known values. Using this procedure, we can get extremely good approximations of probability spaces at a cost related to an a priori known upper bound on the size of the support (i.e., the oracle answer is logarithmic in the size of the support).

5.3. Weak, good, critical, and cocritical conversations. Motivated by the above discussion we make the following definitions.

DEFINITION. *Let* $\epsilon \stackrel{\text{def}}{=} \frac{1}{4t}$, *where* t *is the number of rounds in the interaction* (P,V). *(This setting guarantees that* $(1-\epsilon)^t \geq \frac{3}{4}$.*)*

- *Let* h *be a partial history of the interaction and* β *be a possible next move by the verifier. We say that* β *is* weak *with respect to* h *if*

$$\text{Prob}(V'(h)=\beta) < (1-\epsilon)\cdot\text{Prob}(V(h)=\beta).$$

- *A conversation* $\bar{c}=(c_1,...,c_t)$ *is* i*-weak if* c_i *is weak with respect to* $(c_1,...,c_{i-1})$, *otherwise it is* i-good. (*Note that a conversation can be* i*-weak only if the* i*th move is a verifier move.*)
- *A conversation* $\bar{c}=(c_1,...,c_t)$ *is* i-critical *if it is* i*-weak but* j*-good for every* $j<i$. *A conversation* $\bar{c}$ *is* i-cocritical *if the conversation obtained from* $\bar{c}$, *by complementing (only) the* i*th bit, is* i*-critical.* (*Note that a conversation can be* i*-critical only for a single* i, *yet it may be* i*-cocritical for many* i*'s.*)
- *A conversation is* weak *if it is* i*-weak for some* i, *otherwise it is* good.

We first show that weak conversations occur with negligible probability.

CLAIM 5.2 (rarity of weak conversations). (P',V) *outputs weak conversations with negligible probability.*

Proof. Recall that $[P',V] \stackrel{\text{s}}{=} [P',V']$ and that the same holds also for prefixes of the conversations. Namely, for any $1 \leq i \leq t$, $[P',V]_i \stackrel{\text{s}}{=} [P',V']_i$. Let us define a prefix $h \in \{0,1\}^i$ of a conversation to be *bad* if either

$$\text{Prob}([P',V']_i=h) < \left(1-\frac{\epsilon}{2}\right)\cdot\text{Prob}([P',V]_i=h)$$

or

$$\text{Prob}([P',V']_i=h) > \left(1+\frac{\epsilon}{2}\right)\cdot\text{Prob}([P',V]_i=h).$$

The claim follows by combining two elementary facts.

FACT 5.2.1. *The probability that* (P',V) *outputs a conversation with a bad prefix is negligible.*

Proof. For any $i \leq t$, define B_i to be the set of bad prefixes of length i. By the statistical closeness of $[P',V]_i$ and $[P',V']_i$, we get that

$$\Delta \stackrel{\text{def}}{=} \sum_{h\in B_i} |\text{Prob}([P',V]_i=h) - \text{Prob}([P',V']_i=h)| \;\leq\; \gamma$$

for some negligible fraction γ. On the other hand,

$$\Delta \;=\; \sum_{h\in B_i} \text{Prob}([P',V]_i=h)\cdot\left|1-\frac{\text{Prob}([P',V']_i=h)}{\text{Prob}([P',V]_i=h)}\right| \;>\; \text{Prob}([P',V]_i\in B_i)\cdot\left|\pm\frac{\epsilon}{2}\right|.$$

Thus, $\text{Prob}([P',V]_i \in B_i) < \frac{2\gamma}{\epsilon}$ and the fact follows.

FACT 5.2.2. *If a conversation* $\bar{c}=(c_1,...,c_t)$ *is weak then it contains a bad prefix.*

Proof. Suppose that $\beta \stackrel{\text{def}}{=} c_{i+1}$ is weak with respect $h \stackrel{\text{def}}{=} (c_1,...,c_i)$. If h is a bad prefix then we are done. Otherwise it holds that

$$\text{Prob}([P',V']_i=h) \leq \left(1+\frac{\epsilon}{2}\right)\cdot\text{Prob}([P',V]_i=h).$$

Using the fact that β is weak with respect to h, we get

$$\begin{aligned}\text{Prob}([P',V']_{i+1}=h\circ\beta) &< \left(1+\frac{\epsilon}{2}\right)\cdot(1-\epsilon)\cdot\text{Prob}([P',V]_{i+1}=h\circ\beta)\\ &< \left(1-\frac{\epsilon}{2}\right)\cdot\text{Prob}([P',V]_{i+1}=h\circ\beta),\end{aligned}$$

which implies that $h\circ\beta$ is a bad prefix.

Combining Facts 5.2.1 and 5.2.2, Claim 5.2 follows. □

5.4. Dealing with weak conversations. We start by modifying the prover P', resulting in a modified prover denoted P'', that stops once it gets a verifier message which is weak with respect to the current history; otherwise, P'' behaves as P'.

DEFINITION (modified prover - P''). *For any $h\in\{0,1\}^*$ and $\beta\in\{0,1\}$,*

$$P''(h\circ\beta)=\begin{cases}\text{STOP} & \text{if } \beta \text{ is weak with respect to } h,\\ P'(h\circ\beta) & \text{otherwise.}\end{cases}$$

We assume that the verifier V stops and rejects immediately upon receiving an illegal message from the prover (and in particular upon receiving this STOP *message).*

Next, we modify the simulator, M', so that it outputs either good conversations or truncated conversations which are originally i-critical. Jumping ahead, we stress that such truncated i-critical conversations will be generated from both i-critical and i-cocritical conversations. The modified simulator, denoted M'', proceeds as follows. (We stress that P'' is not necessarily the simulator-based prover of M''.)

DEFINITION (modified simulator - M''). *First, M'' invokes M' and obtains a conversation $\bar{c}=(c_1,...,c_t)$. Next, it queries the augmented oracle on $\bar{c}$. The oracle answers probabilistically and its answers are of the form (i,σ), where $i\in\{1,...,t\}$ and $\sigma\in\{0,1\}$. Finally, M'' halts outputting $(c_1,...,c_{i-1},c_i\oplus\sigma)$.*

In case $\sigma=1$ the output of M'' is NOT a prefix of the output it has obtained from M'. Furthermore, i may be smaller than t, in which case M'' outputs a truncated conversation which, as we see below, is always i-critical; otherwise, M'' outputs a nontruncated conversation. Observe that the oracle message contains $1+\log_2 t$ bits, where t is the length of the interaction between P' and V. It remains to specify the oracle's answer distribution. We first remark that the oracle only returns pairs (i,σ) for which one of the following three conditions holds

1. $\bar{c}$ is good, $i=t$ and $\sigma=0$ (if $\bar{c}$ is good and is not j-cocritical for any j then the oracle always answers this way);
2. $\bar{c}$ is i-critical and $\sigma=0$;
3. $\bar{c}$ is i-cocritical and $\sigma=1$. (Hence, $(c_1,...,c_{i-1},c_i\oplus 1)$ is i-critical.)

To motivate the definition of the augmented oracle, we first consider two special cases. In the first case, the conversation generated by M' is i-critical for some i but is not j-cocritical for any $j<i$. In this case the oracle always answers $(i,0)$ and consequently the simulator always outputs the i-bit long prefix. However, this prefix is still being output with too low probability. This will be corrected by the second case hereby described. In this case, the conversation $\bar{c}$ generated by M' is good and i-cocritical for a single i. This means that the i-bit long prefix is given too much probability weight, whereas the prefix obtained by complementing the ith bit gets too little weight. To correct this, the oracle outputs $(i,1)$ with probability q and $(t,0)$ otherwise, where q will be specified. What happens is that M'' will output the "i-complemented prefix" with higher probability than with which it has appeared in M'. The value of q is determined as follows. Denote $p\stackrel{\text{def}}{=}\text{Prob}(V(c_1,...,c_{i-1})=c_i\oplus 1)$ and

$p' \stackrel{\text{def}}{=} \text{Prob}(V'(c_1,...,c_{i-1})=c_i \oplus 1)$. Then, setting q so that $p'+(1-p')\cdot q = p$ (i.e., $q = \frac{p-p'}{1-p'}$) allows the simulator to output the prefix $(c_1,...,c_{i-1},c_i \oplus 1)$ with the right probability (i.e., as in a (P'',V) interaction). In the general case, the conversation generated by M' may be i-cocritical for many i's as well as j-critical for some (*single*) j. In case it is j-critical, it can be i-cocritical only for $i<j$.

DEFINITION (the augmented oracle answers). *Let us consider the sequence of indices $(i_1,...,i_l)$ for which the generated conversation $\bar{c}$ is critical or cocritical (i.e., the conversation is i_k-cocritical for all $k<l$ and is either i_l-critical or i_l-cocritical). We consider two cases. In both cases the q_k's are set as in the above example; namely, $q_k = \frac{p_k-p'_k}{1-p'_k}$, where $p_k \stackrel{\text{def}}{=} \text{Prob}(V(c_1,...,c_{i_k-1})=c_{i_k}\oplus 1)$ and $p'_k \stackrel{\text{def}}{=} \text{Prob}(V'(c_1,...,c_{i_k-1})=c_{i_k}\oplus 1)$.*

1. *The generated conversation $\bar{c}=(c_1,...,c_t)$ is i_k-cocritical for every $k<l$ and is i_l-critical. In this case, the distribution of the oracle answers is as follows. For every $k<l$, the pair $(i_k,1)$ is returned with probability $(\prod_{j<k}(1-q_j))\cdot q_k$, whereas the pair $(i_l,0)$ appears with probability $\prod_{j<l}(1-q_j)$. We stress that no other pair appears in this distribution (and indeed the reader can easily verify that these probabilities sum up to 1).*
2. *The generated conversation $\bar{c}=(c_1,...,c_t)$ is i_k-cocritical for every $k\le l$. In this case, the distribution of the oracle answers is as follows. For every $k\le l$, the pair $(i_k,1)$ is returned with probability $(\prod_{j<k}(1-q_j))\cdot q_k$, whereas the pair $(t,0)$ appears with probability $\prod_{j\le l}(1-q_j)$. Again, no other pair appears in this distribution. In particular, if $l=0$ then the oracle always returns the pair $(t,0)$.*

CLAIM 5.3 (analysis of the behavior of P'' and M'').

1. $[P'',V] \stackrel{\text{s}}{=} [P',V]$.
2. *Let $\bar{c}$ be an arbitrary conversation of (P'',V). Then*

$$\text{Prob}\,(M''=\bar{c}) \;\ge\; (1-\epsilon)^t \cdot \text{Prob}\,([P'',V]=\bar{c})\,.$$

Recall that $(1-\epsilon)^t \ge \frac{3}{4}$ (by definition of ϵ).

Proof. The weak conversations are negligible in the output distribution of (P',V) (see Claim 5.2). The only difference between $[P'',V]$ and $[P',V]$ originates from a different behavior of P'' on weak conversations; specifically, P'' truncates them while P' does not. Yet, the distribution on the good conversations remains unchanged. Therefore, the distribution of $[P'',V]$ is statistically close to the distribution of $[P',V]$, and we are done with part 1.

We start the proof of part 2 by writing again the probability that (P'',V) outputs $\bar{c}$ as the product of the conditional probabilities of the t steps. Namely,

$$\prod_{i=1}^{t} \text{Prob}\,([P'',V]_{i+1}=h_i\circ c_{i+1} \mid [P'',V]_i=h_i)\,,$$

where $h_i \stackrel{\text{def}}{=} (c_1,...,c_i)$. We do the same for the probability that M'' outputs a conversation $\bar{c}$. We will show by induction that each step of any conversation is produced by M'' with at least $(1-\epsilon)$ times the probability of the same step in the (P'',V)-interaction. Once we have shown this, we are done. Clearly this claim holds for the null prefix. To prove the induction step, we consider the two possibilities for the party making the $i+1$st step.

$i+1$st step is by the prover. Consider the conditional behavior of M'' given the history so far. We will show that this behavior is identical to the behavior of P'' on

the same partial history. A delicate point to note here is that we may talk about the behavior of M'' on a prefix h_i only if this prefix appears with positive probability in the output distribution $[M'']_i$. However, by the induction hypothesis any prefix that is output by $[P'', V]_i$ appears with positive probability in $[M'']_i$.

We partition the analysis into two cases.

1. First, we consider the case in which the last message of the verifier is weak with respect to the history that precedes it. Namely, $h = h' \circ \beta$ and β is weak with respect to h'. In this case, both in the interaction (P'', V) and in the simulation M'', the next message of the prover is set to STOP with probability 1. Namely,

$$\text{Prob}\left(M'' = h \circ \text{STOP} \mid [M'']_i = h\right) \;=\; 1 \;=\; \text{Prob}\left(P''(h) = \text{STOP}\right).$$

2. The other possible case is that the last message of the verifier is not weak with respect to its preceding history. In this case, the simulator M'' behaves like M' and the prover P'' behaves like P'. (Note that the changes in critical and cocritical steps apply only to verifier steps.) Thus,

$$\begin{aligned}\text{Prob}\left([M'']_{i+1} = h \circ \alpha \mid [M'']_i = h\right) \;&=\; \text{Prob}\left([M']_{i+1} = h \circ \alpha \mid [M']_i = h\right)\\ &=\; \text{Prob}\left(P'(h) = \alpha\right)\\ &=\; \text{Prob}\left(P''(h) = \alpha\right).\end{aligned}$$

To summarize, the conditional behavior of M'' in the prover steps and the conditional behavior of P'' are exactly equal.

$i+1$st step is by the verifier. Again, we consider the conditional behavior of M'' given the history so far. Let us recall the modification applied to M' when deriving M''. This modification changes the conditional probability of the verifier steps in the distribution of M' in order to add weight to steps having low probability in the simulation. We note that this modification is made only in critical or cocritical steps of the verifier. Consider a history h_i which might appear in the interaction (P'', V) and a possible response β of V to h_i. Again, by the induction hypothesis, h_i has a positive probability to be output by the simulation M'' and, therefore, we may consider the conditional behavior of M'' on this history h_i. There are three cases to be considered, corresponding to whether either β or $\beta \oplus 1$ or none is weak with respect to h_i.

We start with the simplest case in which neither β nor $\beta \oplus 1$ is weak (w.r.t. h_i). In this case, the behavior of M'' is identical to the behavior of M' since the oracle never sends the message $(i+1, \sigma)$ in this case. However, by the fact that β is not weak, we get that

$$\begin{aligned}(1-\epsilon) \cdot \text{Prob}(V(h) = \beta) \;&\le\; \text{Prob}\left([M']_{i+1} = h \circ \beta \mid [M']_i = h\right)\\ &=\; \text{Prob}\left([M'']_{i+1} = h \circ \beta \mid [M'']_i = h\right)\end{aligned}$$

and we are done with this simple case.

We now turn to the case in which β is weak (w.r.t. h_i). In this case, given that M'' has produced the prefix h_i, it produces $h_i \circ \beta$ whenever M' produces the prefix $h_i \circ \beta$. Furthermore, with conditional probability q (as defined above), M'' produces the prefix $h_i \circ \beta$ also in case M' produces the prefix $h_i \circ (\beta \oplus 1)$. As above, we define

$$\begin{aligned}p \;&\stackrel{\text{def}}{=}\; \text{Prob}\left(V(h_i) = \beta\right),\\ p' \;&\stackrel{\text{def}}{=}\; \text{Prob}\left(V'(h_i) = \beta\right).\end{aligned}$$

Since V' is the simulation-based verifier (for M'), we may also write

$$p' = \text{Prob}\left([M']_{i+1} = h_i \circ \beta \mid [M']_i = h_i\right). \tag{5}$$

Also, recall that q was defined as $\frac{p-p'}{1-p'}$. Now, using these notations,

$$\begin{aligned}\text{Prob}\left([M'']_{i+1} = h_i \circ \beta \mid [M'']_i = h_i\right) &= \text{Prob}\left([M']_{i+1} = h_i \circ \beta \mid [M']_i = h_i\right) \\ &\quad + \frac{p-p'}{1-p'} \cdot \text{Prob}\left([M']_{i+1} = h_i \circ (\beta \oplus 1) \mid [M']_i = h_i\right).\end{aligned}$$

Using equation (5), we get

$$\begin{aligned}&= p' + \frac{p-p'}{1-p'} \cdot (1-p') \\ &= p \\ &= \text{Prob}\left(V(h) = \beta\right).\end{aligned}$$

Finally, we turn to the case in which $\beta \oplus 1$ is weak (w.r.t. h_i). This means that β is cocritical in $\bar{c}$. Given that M'' has produced the prefix h_i, it produces $h_i \circ \beta$ only when M' produces the prefix $h_i \circ \beta$ and, furthermore, M'' does so only with probability $1-q$ (where q is again as defined above). We denote p and p' with respect to the critical message $\beta \oplus 1$. Namely,

$$\begin{aligned}p &\stackrel{\text{def}}{=} \text{Prob}\left(V(h_i) = \beta \oplus 1\right), \\ p' &\stackrel{\text{def}}{=} \text{Prob}\left(V'(h_i) = \beta \oplus 1\right) \\ &= \text{Prob}\left([M']_{i+1} = h_i \circ (\beta \oplus 1) \mid [M']_i = h_i\right).\end{aligned}$$

Thus, recalling that $q = \frac{p-p'}{1-p'}$, we get

$$\begin{aligned}\text{Prob}\left([M'']_{i+1} = h_i \circ \beta \mid [M'']_i = h_i\right) &= \left(1 - \frac{p-p'}{1-p'}\right) \cdot \text{Prob}\left([M']_{i+1} = h_i \circ \beta \mid [M']_i = h_i\right) \\ &= \frac{1-p}{1-p'} \cdot (1-p') \\ &= 1-p \\ &= \text{Prob}\left(V(h_i) = \beta\right).\end{aligned}$$

This completes the proof of Claim 5.3. □

5.5. Lowering the probability of some simulator outputs. By virtue of the modification of M' into M'', we have arrived at a situation in which every conversation appears in the output of M'' with probability which cannot be much smaller than the probability that the conversation appears in $[P'', V]$. Specifically, by part 2 of Claim 5.3 (and by $(1-\epsilon)^t \geq \frac{3}{4}$), we have for every $\bar{c}$

$$\text{Prob}\left(M'' = \bar{c}\right) \geq \frac{3}{4} \cdot \text{Prob}\left([P'', V] = \bar{c}\right). \tag{6}$$

Thus, all that is required is to lower the probabilities that the (modified) simulator outputs each conversation $\bar{c}$ to exactly $\frac{3}{4} \cdot \text{Prob}([P'', V] = \bar{c})$. This can be done by "sieving" the output of M'' using an additional query to the (further-augmented) oracle. Specifically, the modified simulator, denoted M''', runs M'' to obtain a conversation

$\bar{c}$. (Note that M'' always produces output.) Using a further-augmented oracle, M''' outputs $\bar{c}$ with probability

$$p_{\bar{c}} \stackrel{\text{def}}{=} \frac{3}{4} \cdot \frac{\text{Prob}([P'', V] = \bar{c})}{\text{Prob}([M''] = \bar{c})}$$

and halts without output otherwise. Note that $p_{\bar{c}} \leq 1$ holds due to equation (6).

CLAIM 5.4 (analysis of the behavior of M''').

1. *M''' produces output with probability $\frac{3}{4}$.*
2. *The output distribution of M''' (i.e., in case it has output) is identical to the distribution $[P'', V]$.*

Proof. The probability that M''' produces an output is exactly

$$\sum_{\bar{c}} \text{Prob}\left([M''] = \bar{c}\right) \cdot p_{\bar{c}} \;=\; \frac{3}{4}.$$

As for Part (2), we note that the probability that a conversation $\bar{c}$ is output by M''' is exactly $\frac{3}{4} \cdot \text{Prob}\left([P'', V] = \bar{c}\right)$. Since the simulator halts with an output with probability exactly $\frac{3}{4}$, we get that given that M''' halts with an output, it outputs $\bar{c}$ with probability exactly $\text{Prob}\left([P'', V] = \bar{c}\right)$ and we are done. ∎

5.6. Final details. An important point not explicitly addressed so far is whether all the modifications applied to the simulator preserve its ability to be implemented by a probabilistic polynomial-time machine with bounded access to an oracle. Specifically, an issue ignored so far is the ability to efficiently implement the probabilistic choices required of the augmented oracle. A hint toward resolving this problem was given in the technical remark at the end of the motivating subsection (section 5.2). Namely, probabilities of the form $\frac{N}{2^m}$ can be implemented by uniformly selecting a string in $\{0,1\}^m$ and sending it to the oracle which responds with either 0 or 1. However, this only allows to approximate probabilities which are not of the above form. In particular, one can obtain approximations up to exponentially small deviation error. We first comment that such approximations suffice through the entire analysis except for the construction of M''' which must satisfy Claim 5.4 (where the probabilistic behavior must be exact).

Thus, M''' must be implemented with more care. But before we do this, we modify P'' so that it makes its random choices (in case it has any) by flipping a polynomial number of unbiased coins.[2] This modification may change the behavior of P'' slightly, but the deviation can be made so small that the above assertions (specifically Claim 5.3) still hold.

We now turn to the implementation of M'''. Consider the specific "sieving probability" we need to implement when going from M'' to M'''. Namely, $p_{\bar{c}} = \frac{3}{4} \cdot \frac{a/b}{c/d}$, where $\frac{a}{b} = \text{Prob}([P'', V] = \bar{c})$ and $\frac{c}{d} = \text{Prob}([M''] = \bar{c})$. A key observation is that c is the number of coin tosses which lead M'' to output $\bar{c}$. Observing that b is the size of probability space for $[P'', V]$ and using the above modification to P'', we may rewrite $p_{\bar{c}}$ as $\frac{3ad}{4b} \cdot \frac{1}{c} = \frac{e}{c2^f}$, where e and $f = \text{poly}(n)$ are some nonnegative integers.

[2] The implementation of P'' was not discussed explicitly. It is possible that P'' uses an infinite number of coin tosses to select its next message (either 0 or 1). However, an infinite number of coin tosses is not really needed since rounding the probabilities so that a polynomial number of coins suffices, causes only exponentially small rounding errors.

We now note that the oracle can enable the simulator to sieve conversations with probability $\frac{e}{c}$ for any $0 \leq e \leq c$ in the following way. M''' sends to the oracle the random tape ω that it has tossed for M'', and the oracle sieves only e out of the possible c random tapes which lead M'' to output $\bar{c}$. The general case of $p_{\bar{c}} = \frac{e}{c2^f}$ is dealt with by writing $p_{\bar{c}} = \frac{q}{c} + \frac{r}{c2^f}$, where $q = \lfloor \frac{e}{2^f} \rfloor$ and $r = e - q2^f < 2^f$. To implement this sieve, M''' supplies the oracle with a uniformly chosen f-bit long string (in addition to ω). The oracle sieves out q random tapes (of M'') as before and uses the extra bits in order to decide on the sieve in case ω equals a specific (different) random tape. Formally, the process is implemented as follows.

DEFINITION (implementing M''' with an oracle). *Let $f = \mathrm{poly}(n)$. For every possible $\bar{c}$, let $p_{\bar{c}} = \frac{q(\bar{c})}{|\Omega_{\bar{c}}|} + \frac{r(\bar{c})}{2^f \cdot |\Omega_{\bar{c}}|}$, where $\Omega_{\bar{c}}$ is the set of random tapes which makes M'' produce $\bar{c}$ and $0 \leq r(\bar{c}) < 2^f$. Let $G_{\bar{c}} \subseteq \Omega_{\bar{c}}$ be a subset of cardinality $q(\bar{c})$, $a_{\bar{c}} \in \Omega_{\bar{c}} \setminus G_{\bar{c}}$ and $R_{\bar{c}} \subseteq \{0,1\}^f$ be a subset of cardinality $r(\bar{c})$. Then, M''' uniformly selects a random tape ω for M'' and a string $r \in \{0,1\}^f$. Machine M''' queries the oracle on (ω, r) and outputs $M''(\omega)$ if the oracle responds with 1 (otherwise M''' halts with no output). The oracle determines $\bar{c} = M''(\omega)$ and responds 1 if either $\omega \in G_{\bar{c}}$ or $(\omega = a_{\bar{c}}) \wedge (r \in R_{\bar{c}})$, otherwise the oracle responds 0.*

We conclude the proof of Theorem 2 by observing that the following statements hold.

- (P'', V) is an interactive proof system for L. (The completeness condition follows by Claim 5.1 and part 1 of Claim 5.3 which together yield $[P'', V] \stackrel{\mathrm{s}}{=} [P, V]$. Note that we may incur an additional, negligible, completeness error.)
- (P'', V) has *perfect* knowledge complexity $k(n) + 2\log_2 \log_2 n + 2 + \log_2 t = k(n) + O(\log n)$. (The perfectness of the simulator M''' follows by Claim 5.4, whereas the query count follows from the construction: the double-logarithmic term is due to the modification in section 5.1, a $1 + \log_2 t$ term is introduced in the construction of M'', and an additional last query is due to M'''.)

This completes the proof of Theorem 2. □

6. Concluding remarks. We consider our main result as a very first step toward a classification of languages according to the knowledge complexity of their interactive proof systems. In an early version of this paper we suggested two challenges. The first challenge was to provide evidence that NP-complete languages cannot be proven within low (say logarithmic or even constant) knowledge complexity, by showing that languages having logarithmic knowledge complexity are in co$\mathcal{AM}$ (rather than in $\mathcal{BPP}^{\mathcal{NP}}$). Recall that NP is unlikely to be in co$\mathcal{AM}$—see [BHZ-87]. Indeed, following our work and using some of our results, Petrank and Tardos have recently shown that languages having logarithmic knowledge complexity are in $\mathcal{AM} \cap \mathrm{co}\mathcal{AM}$ [PT-96]. The second challenge, still opened, is to try to provide indications that there are languages in $\mathcal{PSPACE}$ which do not have interactive proofs of linear (rather than logarithmic) knowledge complexity. The reader can easily envision more moderate and more ambitious challenges in this direction.

Another interesting question is whether each level of the knowledge-complexity hierarchy contains strictly more languages than previous levels or if some partial collapse occurs. For example, it is open whether the constant knowledge-complexity classes collapse to the zero level.

Regarding our transformation of statistical knowledge complexity into perfect knowledge complexity (i.e., Theorem 2), a few interesting questions arise. First, can the cost of the transformation be reduced to below $O(\log n)$ bits of knowledge? A

result for the special case of statistical zero knowledge will be almost as interesting. Second, can one present an analogous transformation that preserves *one-sided error probability* of the interactive proof? (Note that our transformation introduces a negligible error probability into the completeness condition.) Finally, can one present an analogous transformation that applies to knowledge complexity *with respect to arbitrary verifiers*? (Our transformation applies only to knowledge complexity *with respect to the honest verifier*.)

Appendix A. A flaw in [F-89]. In [F-89], Fortnow presents a constructive method for proving that $\mathcal{SZK} \stackrel{\text{def}}{=} \mathcal{SKC}(0)$ is contained in co-$\mathcal{AM}$. Given an interactive proof (P, V) for a language L and a (statistical) zero-knowledge simulator M (for the honest verifier V), he constructs a two-round protocol (P', V'). This protocol was claimed to constitute an interactive proof system for $\overline{L}$. This claim, as we are going to show, is wrong. Yet, the result $\mathcal{SZK} \subseteq$ co-$\mathcal{AM}$ does hold, since the work of Aiello and Håstad contains the necessary refinements which enable us to present a modified AM-protocol for $\overline{L}$ (see [AH-87, H-94]). Furthermore, Fortnow's basic approach is valid, and indeed it was used in subsequent works (e.g., [AH-87, BMO-90, Ost-91, BP-92, OW-93]).

Fortnow's basic approach starts with the observation that the simulator M must behave differently on $x \in L$ and $x \notin L$. Clearly, the difference cannot be recognized in polynomial time, unless $L \in \mathcal{BPP}$. Yet, stronger recognition devices such as interactive proofs should be able to tell the difference. Fortnow suggests a characterization of the simulator's behavior on $x \in L$ and uses this characterization in his protocol for $\overline{L}$, yet this characterization is wrong. Aiello and Håstad present a refinement of Fortnow's characterization [AH-87]; their characterization is correct and can be used to show that $\mathcal{SZK} \subseteq \mathcal{AM}$ (which is the goal of their paper) as well as $\mathcal{SZK} \subseteq$ co-$\mathcal{AM}$.

Fortnow's characterization. Given an interactive proof (P, V) for L and a simulator M and fixing a common input $x \in \{0,1\}^*$, the following sets are defined. Let us denote by t the number of random bits that the verifier V uses on input x, and by q the number of random bits used by the simulator M. For every conversation prefix h, we consider the set of the verifier's coin tosses which are consistent with h (the conversation so far). We denote this set by R_1^h. Namely, suppose $h = (\alpha_1, \beta_1, ..., \alpha_i, \beta_i)$ or $h = (\alpha_1, \beta_1, ..., \alpha_i, \beta_i, \alpha_{i+1})$. Then, $r \in R_1^h$ iff $V(x, r, \alpha_1, ..., \alpha_j) = \beta_j$ for every $j \leq i$, where $V(x, r, \bar{\alpha})$ denotes the message sent by V on input x random tape r and prover message sequence $\bar{\alpha}$. The set R_1^h depends only on the verifier V. Next, we consider sets R_2^h which are subsets of the corresponding R_1^h's. Specifically, they contain only r's that can appear with h in an accepting conversation output by the simulator M. Namely, $r \in R_2^h$ iff $r \in R_1^h$ and there exists $\omega \in \{0,1\}^q$ so that $M(x, \omega)$ is an accepting conversation with prefix h. (Here $M(x, \omega)$ denotes the conversation output by M on input x and simulator random tape ω.)

Motivation. For simplicity, suppose that the simulation is perfect (i.e., M witnesses that (P, V) is *perfect* zero knowledge) and that (P, V) has one-sided error (i.e., "perfect completeness"). Then, for every $x \in L$ and every possible h, we must have $R_2^h = R_1^h$ (otherwise the simulation is not perfect). However, if $x \notin L$ then there must exist h's so that R_2^h is much smaller than R_1^h. Otherwise the simulator-based prover (for M) will always convince V to accept x, thus violating the soundness condition of (P, V). The problem with the above dichotomy is that it is "too existential" and thus it is not clear how to use it. Instead Fortnow claimed a dichotomy which is more quantitative.

A false characterization. Let $\mathrm{pref}(\bar{c})$ denote the set of all message prefixes in the conversation $\bar{c}$:

- if $x \in L$ then

$$\mathrm{Prob}_\omega(\forall h \in \mathrm{pref}(M(x,\omega)) \ : \ \left|R_2^h\right| \approx_1 \left|R_1^h\right|) \ > \ \frac{3}{4};$$

- if $x \notin L$ then

$$\mathrm{Prob}_\omega(\forall h \in \mathrm{pref}(M(x,\omega)) \ : \ \left|R_2^h\right| \approx_2 \left|R_1^h\right|) \ < \ \frac{1}{4},$$

where the probability (in both cases) is taken uniformly over $\omega \in \{0,1\}^q$. We did not specify what is meant by $\approx_i$. One may substitute $\alpha \approx_1 \beta$ by $\alpha \geq \frac{1}{2} \cdot \beta$ and $\alpha \approx_2 \beta$ by $\alpha \geq \frac{1}{4} \cdot \beta$. The gap between the two is needed for the approximate lower/upper bound protocols.

A counterexample. The mistake is in the second item of the characterization. The false argument given in [F-89] confuses between the probability distribution of conversations output by the simulator and the probability distribution of the conversations between a simulator-based prover (denote P^*) and the verifier. These distributions are not necessarily the same (note that we are in the case $x \notin L$). Consequently, the probability that "good" conversations (i.e., conversations for which $|R_2| \approx |R_1|$ for all prefixes) occur in the (P^*, V) interaction is not the same as the probability that the simulator outputs "good" conversations. This point is ignored in [F-89] and leads there to the false conclusion that the characterization holds. Below, we present an interactive proof (P, V) and a (perfect) zero-knowledge simulator for which the characterization fails.

The interactive proof that we present is for the empty language Φ. This interactive proof is perfect zero knowledge for the trivial reason that the requirement is vacuous. Yet, we present a simulator for this interactive proof which, for every $x \in \{0,1\}^* = \overline{\Phi}$, outputs "good" conversation with probability close to 1. Thus, the characterization fails.

The interactive proof (from the verifier's point of view – input $x \in \{0,1\}^n$).

- The verifier uniformly selects $\alpha \in \{0,1\}^n$ and sends α to the prover.
- The verifier waits for the prover's message $\beta \in \{0,1\}^n$.
- Next, the verifier uniformly selects $\gamma \in \{0,1\}^n$ and sends γ to the prover.
- The verifier accepts iff either $\alpha = 0^n$ or $\beta = \gamma$.

Regardless of the prover's strategy, the verifier accepts each $x \in \{0,1\}^n$ with negligible probability; specifically, $2^{-n} + (1 - 2^{-n}) \cdot 2^{-n}$. Thus, the above protocol indeed constitutes an interactive proof for the empty language Φ.

The simulator operates as follows (on input $x \in \{0,1\}^n$ and parameter ϵ).

- With probability $1 - \epsilon$, the simulator M outputs a conversation uniformly distributed in $0^n \times \{0,1\}^{2n}$.
- With probability ϵ, the simulator M outputs a conversation uniformly distributed in $(\{0,1\}^n - 0^n) \times \{0,1\}^{2n}$.

The parameter ϵ is set to be negligible, say, $\epsilon = 2^{-n}$.

Claim. In contradiction to the characterization, for every $x \in \{0,1\}^* = \overline{\Phi}$,

$$\mathrm{Prob}_\omega(\forall h \in \mathrm{pref}(M(x,\omega)) \ : \ \left|R_2^h\right| = \left|R_1^h\right|) \ \geq \ 1 - \epsilon,$$

where the probability is taken uniformly over $\omega \in \{0,1\}^q$.

Proof. Recall that all conversations are $3n$-bit long strings and for a conversation $\alpha\beta\gamma \in \{0,1\}^{3n}$ the verifier coins are $\alpha\gamma$. Note that with probability $1-\epsilon$, the simulator outputs a conversation of the form $0^n\beta\gamma$. Thus, it suffices to show that every conversation of the form $0^n\beta\gamma$ satisfies $R_2^h = R_1^h$ for each prefix (i.e., $h \in \{\lambda, 0^n, 0^n\beta, 0^n\beta\gamma\}$). First observe that $R_1^\lambda = \{0,1\}^{2n} = R_2^\lambda$, since for every $\alpha\gamma \in \{0,1\}^{2n}$ the simulator outputs the accepting conversation $\alpha\gamma\gamma$ with nonzero probability. Similarly, $R_1^{0^n} = 0^n\{0,1\}^n = R_2^{0^n}$ (here we use $\alpha = 0^n$). Next, for every $\beta \in \{0,1\}^n$ we have $R_1^{0^n\beta} = 0^n\{0,1\}^n = R_2^{0^n\beta}$, since for every $\gamma \in \{0,1\}^n$ the simulator outputs the accepting conversation $0^n\beta\gamma$ with nonzero probability. (Here we use the fact that the verifier always accepts when $\alpha = 0^n$.) Similarly, $R_1^{0^n\beta\gamma} = 0^n\gamma = R_2^{0^n\beta\gamma}$. □

Conclusion. The source of trouble is that the definition of the set R_2^h does not take into account the probability weight assigned by the simulator to ω's that witness the assertion "the simulator outputs an accepting conversation that starts with h." Indeed, this is exactly the nature of the refinement suggested by Aiello and Håstad [AH-87].

Appendix B. Interactive proofs with nonnegligible error probabilities. As explained in Remark 1 of section 3.1, the notion of an interactive proof with bounded knowledge complexity is not robust under changes in the allowed error probability. Throughout the paper, we use the natural definition of interactive proofs in which the error probability is negligible. However, our techniques yield nontrivial results also in the case where one defines interactive proofs with some specific nonnegligible error probability. In this appendix we explain how such assertions may be obtained and state such results for two special cases.

Denote by $\epsilon_c(n)$ an upper bound on the probability that the verifier rejects an input x although $x \in L$ and the prover plays honestly. This is the error probability related to the completeness condition. Similarly, denote by $\epsilon_s(n)$ an upper bound on the probability that the verifier accepts $x \notin L$ when the prover follows its optimal strategy (not necessarily following the protocol). This is the error probability related to the soundness condition. We say that an interactive proof has error probabilities (ϵ_s, ϵ_c) if its error probability in the soundness condition is bounded by ϵ_s and its error probability in the completeness condition is bounded by ϵ_c.

B.1. The perfect case. In this subsection, we consider the restricted case of *perfect* knowledge complexity and derive Theorem B.1 which is the analogue of Theorem 1 for the case that the error probabilities are not negligible. Following the definitions in section 4, we denote the simulation-based prover by P^*.

Let us follows the steps of the proof of our main theorem and observe which assertions hold for the case of nonnegligible error probability. We begin by observing that the following generalization of Lemma 4.2 holds.

LEMMA B.1. *Let (P,V) be an interactive proof for L with error probabilities $(\epsilon_s(n), \epsilon_c(n))$ and with knowledge complexity $k(n)$, then the following hold.*

1. *If $x \in L$ then the probability that (P^*,V) outputs an accepting conversation is at least $(1-\epsilon_c(n))^2 \cdot 2^{-k(n)}$, where $n = |x|$.*
2. *If $x \notin L$ then the probability that (P^*,V) outputs an accepting conversation is at most $\epsilon_s(n)$, where $n = |x|$.*

The proof of this lemma is identical to the proof of Lemma 4.2, except that here $\frac{|A_\lambda|}{|S_\lambda|} \geq 1 - \epsilon_c(n)$. As explained in section 4, an efficient machine with access to an NP-oracle can sample conversations in (P^*,V). By Lemma B.1, this would yield an

accepting conversation with probability at most $\epsilon_s(n)$ in the case $x \notin L$ and at least $(1-\epsilon_c(n))^2 \cdot 2^{-k(n)}$ when $x \in L$. In case these two probabilities differ sufficiently (i.e., by more then a polynomial fraction), we can use standard amplification techniques to get a probabilistic algorithm that determines whether $x \in L$ with error probability less than $\frac{1}{3}$ (or negligible, or 2^{-n}). To summarize, we get the following theorem for perfect knowledge complexity.

THEOREM B.1. *If a language L has an interactive proof with perfect knowledge complexity $k(n)$ and error probabilities (ϵ_s, ϵ_c) and if there exists a positive polynomial $p(n)$ such that*

$$(1-\epsilon_c(n))^2 \cdot 2^{-k(n)} \;>\; \epsilon_s(n) + \frac{1}{p(n)},$$

then $L \in \mathcal{BPP}^{\mathcal{NP}}$.

Examples. Theorem B.1 implies, for example, that if a language L has an interactive proof of knowledge complexity 1 and error probability $\frac{1}{4}$ (both in the soundness condition and in the completeness condition), then L is in $\mathcal{BPP}^{\mathcal{NP}}$. Another interesting example is the case of one-sided error (i.e., $\epsilon_c = 0$). Theorem B.1 implies that, for any polynomial $p(\cdot)$, if a language L has a one-sided error interactive proof (P, V) of knowledge-complexity at most $\log_2 p(\cdot)$ and error probability $\epsilon_s \leq \frac{1}{2p(\cdot)}$, then L is in $\mathcal{BPP}^{\mathcal{NP}}$.

B.2. The general (statistical) case. Unfortunately, the analogue result for statistical knowledge complexity is not as clean and has various different formulations according to possible properties of the error probabilities. Let us explain how such results can be obtained and give a specific example for the special case in which $\epsilon_c = 0$; i.e., the original interaction has one-sided error.

Recall that the proof for the negligible error-probability case uses the transformation from statistical to perfect knowledge complexity (i.e., Theorem 2) and then uses Theorem 1. This transformation increases the knowledge complexity by a logarithmic additive term. In view of Lemma B.1, it is desirable not to increase the knowledge complexity without concurrently decreasing the error probability. Thus, before applying the transformation, we reduce the error probability by iterating the protocol as many times as possible while maintaining logarithmic knowledge complexity.

Specifically, we start with a protocol (P, V) of statistical knowledge complexity $k(\cdot)$ and denote by $l(\cdot)$ the total length of the conversation in this protocol. Also, fix an input x of length n, and let $l = l(n)$, $k = k(n)$, $\epsilon_s = \epsilon_s(n)$, and $\epsilon_c = \epsilon_c(n)$. We begin by running the original protocol (P, V) sequentially $t \stackrel{\text{def}}{=} \lceil (\log_2 l)/k \rceil$ times. These repetitions yield a new protocol (P', V') whose length is $t \cdot l$, its knowledge-complexity is bounded by $t \cdot k < k + \log_2 l$, and its error probability decreases. To compute the decrease in the error probabilities, we partition the analysis into two cases according to whether the original protocol has one-sided error or not.

If the original interaction has one-sided error, i.e., the verifier always accepts when $x \in L$, then the new verifier V' accepts only if ALL repetitions of the original protocols end up accepting. The error probabilities in this case decrease from $(\epsilon_s, 0)$ to $(\epsilon_s^t, 0)$. In the case where the original interactive proof was not one sided, the verifier counts the number of original interactions that end up with the original verifier accepting. The new verifier accepts if this number is greater than $\frac{\epsilon_s + (1-\epsilon_c)}{2} \cdot t$. In order to compute the new error probabilities we may apply the Chernoff bound and get an upper bound on the new error probabilities which depends on t, on the difference between $1 - \epsilon_c$ and ϵ_s, and, of course, on ϵ_s and ϵ_c themselves.

Next, we apply the transformation of section 5 and get a new interactive proof (P'', V'') for L which has knowledge complexity $(k + \log l) + 2 + 2\log_2 \log_2 n + \lceil \log_2(l \cdot t) \rceil$, where the additional $2 + 2\log_2 \log_2 n + \lceil \log_2(l \cdot t) \rceil$ term comes from the transformation. Finally, if the resulting parameters of (P'', V'') satisfy the conditions stated in Theorem B.1, then we get that the language L is in $\mathcal{BPP}^{\mathcal{NP}}$. Let us provide full details for the special (yet important) case of one-sided error (i.e., $\epsilon_c = 0$).

In the special case of one-sided error, we end up using Theorem B.1 for an interactive proof with knowledge complexity $(k + \log l) + 2 + 2\log_2 \log_2 n + \lceil \log_2(l \cdot t) \rceil$ and error probabilities $({\epsilon_s}^t, \epsilon)$, where ϵ is a negligible fraction (introduced by the transformation). Thus, we get the following theorem for statistical knowledge complexity.

THEOREM B.2. *Suppose that a language L has an interactive proof of statistical knowledge complexity $k(n)$, one-sided error probability $\epsilon_s(n)$, and with length $l(n)$ so that there exists a polynomial $p(n)$ for which the following inequality holds:*

$$\frac{1}{8 \cdot (\log_2 n)^2 \cdot 2^{k(n)} \cdot l(n)^2 \cdot \left\lceil \frac{\log_2 l(n)}{k(n)} \right\rceil} \geq \epsilon_s(n)^{\lceil (\log_2 l(n))/k(n) \rceil} + \frac{1}{p(n)}.$$

Then $L \in \mathcal{BPP}^{\mathcal{NP}}$.

For $l(n) \leq 2^{k(n)}$ the condition simplifies to $2^{-3k(n)} \geq 8(\log_2 n)^2 \cdot \epsilon_s(n) + 1/\mathrm{poly}(n)$, whereas for $l(n) > 2^{k(n)}$ the condition simplifies to

$$\frac{1}{8 \cdot (\log_2 n)^3 \cdot l(n)^3} \geq \epsilon_s(n)^{\lceil (\log_2 l(n))/k(n) \rceil} + \frac{1}{\mathrm{poly}(n)}.$$

Acknowledgment. We thank Leonard Shulman for providing us with a simpler proof of Claim 4.3.

REFERENCES

[ABV-95] W. AIELLO, M. BELLARE, AND R. VENKATESAN, *Knowledge on the average – Perfect, statistical and logarithmic*, in Proceedings of the 27th Annual ACM Symposium on the Theory of Computing, Association for Computing Machinery, New York, 1995.

[AH-87] W. AIELLO AND J. HÅSTAD, *Perfect zero-knowledge can be recognized in two rounds*, in Proc. 28th Annual IEEE Symposium on the Foundations of Computer Science, IEEE, Piscataway, NJ, 1987.

[B-85] L. BABAI, *Trading group theory for randomness*, in Proc. 17th Annual ACM Symposium on the Theory of Computing, Association for Computing Machinery, New York, 1985.

[BM-88] L. BABAI AND S. MORAN, *Arthur-Merlin games: A randomized proof system and a hierarchy of complexity classes*, J. Comput. System Sci., 36 (1988), pp. 254–276.

[BMO-90] M. BELLARE, S. MICALI, AND R. OSTROVSKY, *The (true) complexity of statistical zero-knowledge*, in Proc. 22nd Annual ACM Symposium on the Theory of Computing, Association for Computing Machinery, New York, 1990.

[BP-92] M. BELLARE AND E. PETRANK, *Making zero-knowledge provers efficient*, in Proc. 24th Annual ACM Symposium on the Theory of Computing, Association for Computing Machinery, New York, 1992.

[B+ 88] M. BEN-OR, S. GOLDWASSER, O. GOLDREICH, J. HÅSTAD, J. KILIAN, S. MICALI, AND P. ROGAWAY, *Everything provable is provable in zero-knowledge*, Advances in Cryptology—Proceedings of CRYPTO 88, S. Goldwasser, ed., Lecture Notes in Computer Science 403, Springer-Verlag, Berlin, New York, 1989.

[BHZ-87] R. BOPPANA, J. HÅSTAD, AND S. ZACHOS, *Does co-NP have short interactive proofs*, Inform. Process. Lett., 25 (1987), pp. 127–132.

[F-89] L. FORTNOW, *The complexity of perfect zero-knowledge*, Adv. Comput. Res., 5 (1989), pp. 327–343.

[GMS-87] O. Goldreich, Y. Mansour, and M. Sipser, *Interactive proof systems: Provers that never fail and random selection*, in Proc. 28th Annual IEEE Symposium on the Foundations of Computer Science, IEEE, Piscataway, NJ, 1987.

[GMW-86] O. Goldreich, S. Micali, and A. Wigderson, *Proofs that yield nothing but their validity and a methodology of cryptographic protocol design*, J. Assoc. Comput. Mach., 38 (1991), pp. 691–729.

[GP-91] O. Goldreich and E. Petrank, *Quantifying knowledge complexity*, in Proc. 32nd Annual IEEE Symposium on the Foundations of Computer Science, IEEE, Piscataway, NJ, 1991; Comput. Complexity, to appear.

[GMR-85] S. Goldwasser, S. Micali, and C. Rackoff, *The knowledge complexity of interactive proofs*, in Proc. 17th Annual ACM Symposium on the Theory of Computing, Association for Computing Machinery, New York, 1985.

[GMR-89] S. Goldwasser, S. Micali, and C. Rackoff, *The knowledge complexity of interactive proofs*, SIAM J. Comput., 18 (1989), pp. 186–208.

[GS-89] S. Goldwasser and M. Sipser, *Private coins vs. public coins in interactive proof systems*, Adv. Comput. Res., 5 (1989), pp. 73–90.

[H-94] J. Håstad, *Perfect zero-knowledge in $\mathcal{AM} \cap$ co-$\mathcal{AM}$*, 1994, manuscript. Explains the underlying ideas behind [AH-87].

[IY-87] R. Impagliazzo and M. Yung, *Direct minimum-knowledge computations*, in Advances in Cryptology—Proceedings of CRYPTO 87, Lecture Notes in Computer Science 293, Springer-Verlag, Berlin, New York, 1987.

[JVV-86] M. Jerrum, L. Valiant, and V. Vazirani, *Random generation of combinatorial structures from a uniform distribution*, Theoret. Comput. Sci., 43 (1986), pp. 169–188.

[LFKN-90] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, *Algebraic methods for interactive proof systems*, in Proc. 31st Annual IEEE Symposium on the Foundations of Computer Science, IEEE, Piscataway, NJ, 1990, pp. 2–10.

[Ost-91] R. Ostrovsky, *One-way functions, hard on average problems, and statistical zero-knowledge proofs*, in Proc. Structures in Complexity Theory 6th Annual Conference IEEE, Piscataway, NJ, 1991.

[OW-93] R. Ostrovsky and A. Wigderson, *One-way functions are essential for non-trivial zero-knowledge*, in Proc. 2nd Israeli Symp. on Theory of Computing and Systems, IEEE, Piscataway, NJ, 1993, pp. 3–17.

[PT-96] E. Petrank and G. Tardos, *On the knowledge complexity of NP*, in Proc. 37th IEEE Symposium on the Foundations of Computer Science, IEEE, Piscataway, NJ, 1996, pp. 494–503.

[Sh-90] A. Shamir, *IP=PSPACE*, in Proc. 31st Annual IEEE Symposium on the Foundations of Computer Science, IEEE, Piscataway, NJ, 1990, pp. 11–15.

[Si-83] M. Sipser, *A complexity theoretic approach to randomness*, in Proc. 15th Annual ACM Symposium on the Theory of Computing, Association for Computing Machinery, New York, 1983.

[St-83] L. Stockmeyer, *The complexity of approximate counting*, in Proc. 15th Annual ACM Symposium on the Theory of Computing, Association for Computing Machinery, New York, 1983.

SIAM J. COMPUT.
Vol. 27, No. 4, pp. 1142–1167, August 1998
© 1998 Society for Industrial and Applied Mathematics
012

THE COMPLEXITY OF PLANAR COUNTING PROBLEMS*

HARRY B. HUNT III†, MADHAV V. MARATHE‡, VENKATESH RADHAKRISHNAN§, AND RICHARD E. STEARNS†

Abstract. We prove the #P-hardness of the counting problems associated with various satisfiability, graph, and combinatorial problems, when restricted to planar instances. These problems include 3SAT, 1-3SAT, 1-EX3SAT, MINIMUM VERTEX COVER, MINIMUM DOMINATING SET, MINIMUM FEEDBACK VERTEX SET, X3C, PARTITION INTO TRIANGLES, AND CLIQUE COVER. We also prove the NP-completeness of the AMBIGUOUS SATISFIABILITY problems [J. B. Saxe, *Two Papers on Graph Embedding Problems*, Tech. Report CMU-CS-80-102, Dept. of Computer Science, Carnegie Mellon Univ., Pittsburgh, PA, 1980] and the D^P-completeness (with respect to random polynomial reducibility) of the unique satisfiability problems [L. G. Valiant and V. V. Vazirani, *NP is as easy as detecting unique solutions*, in Proc. 17th ACM Symp. on Theory of Computing, 1985, pp. 458–463] associated with several of the above problems, when restricted to planar instances. Previously, very few #P-hardness results, no NP-hardness results, and no D^P-completeness results were known for counting problems, ambiguous satisfiability problems, and unique satisfiability problems, respectively, when restricted to planar instances.

Assuming $\mathsf{P} \neq \mathsf{NP}$, one corollary of the above results is that there are no ϵ-approximation algorithms for the problems of maximizing or minimizing a linear objective function subject to a planar system of linear inequality constraints over the integers.

Key words. planar, 3SAT, graphs, #P-complete, D^P-complete, NP-complete

AMS subject classifications. 68Q15, 68R10.

PII. S0097539793304601

1. Introduction.

A number of papers in the literature, including [1, 5, 16, 20, 21, 25, 26], have considered the complexity of counting problems, proving many such problems to be #P-complete. Other papers have studied the complexity of ambiguous and unique satisfiability problems [23, 27], proving such problems to be NP-hard and D^P-hard,[1] respectively. Still other papers [3, 4, 5, 6, 15] have considered the complexity of decision problems when restricted to planar instances, proving many such problems to be NP-hard. In this paper, we combine these lines of research and prove for the first time that *even when restricted to planar instances*, many counting problems remain #P-complete, many ambiguous satisfiability problems remain NP-complete, and many unique satisfiability problems remain D^P-hard.

Previously, very few #P-hardness results, no NP-hardness results, and no D^P-completeness results were known for counting problems, ambiguous satisfiability problems, and unique satisfiability problems, respectively, when restricted to planar in-

*Received by the editors August 15, 1993; accepted for publication (in revised form) June 12, 1996; published electronically May 19, 1998.

http://www.siam.org/journals/sicomp/27-4/30460.html

†Department of Computer Science, State University of New York (SUNY) at Albany, Albany, NY 12222 (hunt@cs.albany.edu, res@cs.albany.edu). This work was supported by NSF grants CCR 89-03319, CCR 94-06611, and CCR 90-06396.

‡Los Alamos National Laboratory, P.O. Box 1663, MS B265, Los Alamos, NM 87545 (marathe@lanl.gov). This research was supported by the Department of Energy under contract W-7405-ENG-36. Part of the work was done while the author was at SUNY-Albany and supported by NSF grants CCR 89-03319 and CCR 94-06611.

§Mailstop 47LA-2, Hewlett-Packard Company, 19447 Pruneridge Avenue, Cupertino, CA 95014-9913 (rven@cup.hp.com). Part of this work was done while the author was at SUNY-Albany and was supported by NSF grants CCR 89-03319 and CCR 94-06611.

[1]Throughout this paper, all the D^P-hardness results are with respect to random polynomial time reductions.

stances. Such results are presented for several satisfiability, graph, and combinatorial problems. These results show that planar counting, planar ambiguous satisfiability, and planar unique satisfiability problems are *as hard as* arbitrary such problems, with respect to polynomial time and random polynomial time reducibilities. The results in this paper both extend the results in the literature and also provide additional tools for proving hardness results for planar problems for various complexity classes. These tools include parsimonious and weakly parsimonious crossover boxes, the NP-hardness of various basic planar satisfiability problems, and the NP-hardness of the planar ambiguous satisfiability problems. (Henceforth, we denote the restriction of a problem Π to planar instances by PL-Π.) The particular results presented here include the following.

1. The problem 3SAT has a parsimonious planar crossover box. Among other things, this implies that the problem #PL-3SAT is #P-complete, the problem AMBIGUOUS-PL-3SAT is NP-complete, and the problem UNIQUE-PL-3SAT is D^P-complete.

2. The problem 3SAT is simultaneously polynomial time, planarity-preserving, and parsimoniously reducible to each of the basic CNF satisfiability problems listed in Table 1.1. (Previously, 3SAT was only known to be so reducible to the problem 1-EX3SAT [3].)

3. There exist polynomial time, weakly parsimonious, and planarity-preserving reductions from the problem 1-EX3MONOSAT to several graph problems including MINIMUM VERTEX COVER, MINIMUM DOMINATING SET, and MINIMUM FEEDBACK VERTEX SET.

4. Using the results 1, 2, and 3, variants of known reductions, and new reductions, we show that for all of the problems PL-Π in Table 1.1, the problem #PL-Π is #P-hard. Similarly, we show that for many of the problems in Table 1.1, the problems AMBIGUOUS PL-Π and UNIQUE PL-Π are NP-hard and D^P-hard, respectively.

5. All of the #P-hardness results for planar counting problems in Table 1.1 can easily be shown to hold, even when the input to the problem consists of a formula f and one of its satisfying assignments, a graph G and one of its minimum vertex covers, a graph G and one of its dominating sets, etc.

Thus the #P-hardness of the problems #PL-Π in Table 1.1 is not simply a corollary of the NP-hardness of the problem PL-Π, since the problems #PL-Π are #P-hard even when restricted to sets of instances for which the problems PL-Π are trivially polynomial time solvable. Moreover, quoting from [27] we note that "whether the number of solutions of all NP-complete problems are nevertheless polynomial time interreducible (i.e., whether NP-completeness implies #P-completeness) is still open."

Corollaries of our results and their proofs include the following.

1. The problems #1-VALID 3SAT and #1-VALID PL-3SAT are #P-complete. (It is trivially seen that every instance of the problem 1-VALID 3SAT is satisfiable by the assignment making all variables equal to 1. See Definition 2.10.)

2. The problems AMBIGUOUS 1-VALID 3SAT and AMBIGUOUS 1-VALID PL-3SAT are NP-complete. The problems UNIQUE 1-VALID 3SAT and UNIQUE 1-VALID PL-3SAT are Co-NP-complete.

3. Assuming $P \neq NP$, there are no ϵ-approximation algorithms for the problems of maximizing or minimizing a linear objective function subject to a planar system of inequalities over the integers.

Table 1.1 gives a summary of our #P-hardness results. The rest of the paper is organized as follows. Section 2 contains definitions and preliminaries. Section 3

discusses the complexity of #Pl-3Sat and other basic CNF satisfiability problems. These problems are used to prove the #P-hardness of other problems discussed in the subsequent sections. Section 4 discusses the complexity of various counting problems for planar graphs. Section 5 contains the ambiguous and unique satisfiability results and the result on the nonapproximability of the objective functions of integer linear programs. Finally, section 6 consists of conclusions and open problems.

Table 1.1

Summary of NP- and #P-hardness results for planar instances. The third column summarizes the decision complexity of the problems while the fourth column summarizes the complexity of the counting versions. A star (*) denotes a result obtained in this paper. The numbers in square brackets are the references where the corresponding results are proved.

No.	Problem	Decision problem (NP-complete)	Counting version #P-hard
1	Pl-3Sat	[15]	*
2	Pl-Ex3Sat	*	*
3	Pl-1-3Sat	[3]	*
4	Pl-1-Ex3Sat	[3]	*
5	Pl-1-Ex3MonoSat	*	*
6	Pl-Vertex Cover	[6]	*
7	Pl-Hamiltonian Circuit	[15]	[20]
8	Pl-Dominating Set	[6]	*
9	Pl-Feedback Vertex Set	[6]	*
10	Pl-3-Coloring	[6]	[11]
11	Pl-Graph Homomorphism and Onto Homomorphism	[5]	[11]
12	Pl-Subgraph Isomorphism	[5]	*
13	Pl-Clique Cover	[3]	*
14	Pl-Hitting Set	[3]	*
15	Pl-X3C	[3]	*
16	Pl-Partition Into Triangles	[3]	*
17	Pl-Partition Into Claws	[3]	*

2. Definitions and preliminaries. In this section we review the basic definitions and notation used in this paper. Additional definitions can be found in [3, 5, 19, 23].

Definition 2.1. *A* search problem Π *consists of a set* D_Π *of objects called* instances, *and for each instance* $I \in D_\Pi$, *a set* $S_\Pi[I]$ *of objects called* solutions *for* I. *An algorithm is said to* solve *a search problem* Π *if, given* $I \in D_\Pi$ *as input, the algorithm outputs* no *if* $S_\Pi[I] = \phi$ *and outputs an* $s \in S_\Pi[I]$ *otherwise.*

Definition 2.2. *The* enumeration problem *associated with a search problem* Π *is the problem of determining, given* $I \in D_\Pi$, *the cardinality of* $S_\Pi[I]$.

Definition 2.3. *The class* #P *consists of all enumeration problems associated with search problems* Π *such that there is a nondeterministic algorithm for solving* Π *such that, for all* $I \in D_\Pi$, *the number of distinct accepting sequences for* I *by the algorithm equals* $|S_\Pi[I]|$ *and the length of the longest accepting computation of the algorithm on* $I \in D_\Pi$ *is bounded by a polynomial in the length of* I.

Definition 2.4. *A reduction* [5] $f : D_\Pi \to D_{\Pi'}$ *is parsimonious if and only if* $\forall I \in D_\Pi$ *the number of solutions of* I *is equal to the number of solutions of* $f(I)$.

Definition 2.5. *A reduction* f *is weakly parsimonious if and only if* $|S(I)| = g(I)|S(f(I))|$, *where* $|S(I)|$ *and* $|S(f(I))|$ *denote the number of solutions of* I *and* $f(I)$, *respectively, and* $g(I)$ *is a polynomial time computable function represented using binary notation.*

An enumeration problem is said to be #P-hard if each problem in #P is polynomial time parsimoniously or weakly parsimoniously Turing reducible to it. If, in addition, the enumeration problem is in #P, the problem is said to be #P-complete.

DEFINITION 2.6. #3SAT *is the problem of computing the number of satisfying assignments of a Boolean formula F in conjunctive normal form with at most three literals per clause.*

The following basic results on the complexity of counting problems are used in this paper.

THEOREM 2.7 (see [5, 25]). *The problems* #3SAT *and* #GRAPH 3-COLORING *are* #P*-complete.*

DEFINITION 2.8. *The* bipartite graph *associated with a CNF satisfiability problem is defined as follows. The clauses and variables in a formula are in one to one correspondence with the vertices of the graph. There is an edge between a clause node and a variable node if and only if the variable appears in the clause. A CNF formula is planar if and only if its associated bipartite graph is planar.*

DEFINITION 2.9. (1) EX3SAT *is the restriction of the problem* 3SAT *to formulas in which each clause has exactly three literals.*

(2) 1-3SAT *is the problem of determining if a CNF formula in which each clause has no more than three literals has a satisfying assignment such that exactly one literal per clause is satisfied.*

(3) 1-EX3SAT *is the problem of determining if a CNF formula in which each clause has exactly three literals has a satisfying assignment such that exactly one literal per clause is satisfied.*

(4) 1-EX3MONOSAT *is the restriction of* 1-EX3SAT *to formulas having no negated literals.*

DEFINITION 2.10 (see [24]). *A relation $R(x_1, x_2, \ldots, x_n)$ is* 1-valid *if and only if $(1, 1, \ldots, 1) \in R$. A CNF formula f is* 1-valid *if the formula is satisfied when all the variables in the formula are set to true.*

DEFINITION 2.11. *Given a Boolean formula F and an assignment $\mathbf{v}$ to the variables of F, the notation $\mathbf{v}[F]$ denotes the value of F under $\mathbf{v}$.*

DEFINITION 2.12. (1) EXACT COVER BY 3-SETS (X3C): *An instance of this problem consists of a set X with $3m$ elements and a collection C of three-element subsets of X. The question is: does there exist a subcollection C' of C such that every element of X occurs in exactly one set in C'?*

(2) HITTING SET: *An instance of this problem consists of a collection C of subsets of a finite set S and a positive integer $K \leq |C|$. The question is: is there a subset $S' \subseteq S$ with $|S'| \leq K$ such that S' contains at least one element from each subset in C?*

As in the case of 3SAT, we can associate a bipartite graph $G = (S, T, E)$ with an instance of each of the above problems. For example, PL-X3C is defined as follows. Each element in C has a corresponding vertex in S, each element in X has a corresponding vertex in T, and a vertex in S is joined to a vertex in T if and only if the set corresponding to the vertex in S contains the element corresponding to the vertex in T.

DEFINITION 2.13. (1) DOMINATING SET: *An instance of this problem consists of an undirected graph $G = (V, E)$ and an integer $K \leq |V|$. The question is: is there a dominating set of size at most K in G; i.e., is there a subset V' of V, $|V'| \leq K$, such that for each $u \in V - V'$ there is a $v \in V'$ such that $(u, v) \in E$?*

(2) CLIQUE COVER: *An instance of this problem consists of an undirected graph $G = (V, E)$ and an integer $K \leq |V|$. The question is: is there a clique cover of size*

at most K in G; i.e., can the graph be partitioned into at most K sets of nodes such that each set is a clique?

(3) PARTITION INTO CLAWS: *An instance of this problem consists of an undirected graph $G = (V, E)$, $|E| = m$. The question is: is there a way to partition the edges of the graph into sets $E_1, \ldots, E_s$, $s = m/3$, such that each E_i induces a subgraph isomorphic to $K_{1,3}$ (i.e., a claw)?*

(4) FEEDBACK VERTEX SET: *An instance of this problem consists of a directed graph $G = (V, E)$ and an integer $K \leq |V|$. The question is: is there a feedback vertex set of size at most K in G; i.e., does there exist a subset V' of V of size at most K such that V' contains at least one vertex from every cycle in G?*

DEFINITION 2.14. *Let Π denote a CNF satisfiability problem. Then the associated* ambiguous version *of Π, denoted by* AMBIGUOUS-Π, *is the problem of determining, given an instance I of Π and an assignment $\mathbf{v}$ to the variables of I such that $\mathbf{v}[I] = 1$, if there is an additional assignment of values to the variables of I satisfying I. The associated* unique version *of Π, denoted by* UNIQUE-Π, *is the problem of determining if the given instance I of Π has exactly one satisfying assignment.*

More generally, let Π be any decision problem. Henceforth, when applicable, we denote the restriction of Π to planar instances by PL-Π, the counting version of Π by #Π, the ambiguous version of Π by AMBIGUOUS-Π, and the unique version of the problem by UNIQUE-Π. For example, recalling Definition 2.12 and the discussion immediately following it,

(i) PL-X3C is the problem X3C restricted to instances (X, C) for which the bipartite graph $G = (S, T, E)$ is planar;

(ii) #X3C is the problem of computing, given (X, C), the number of distinct subsets $C' \subseteq C$ such that each element of X occurs in exactly one set in C';

(iii) AMBIGUOUS-X3C is the problem of determining, given (X, C, C'), where $C' \subseteq C$ and each element of X occurs in exactly one set in C', if there exists another subset $C'' \subseteq C$ that is an exact cover of the elements in X.

Finally, henceforth, by *reduction*, we mean a polynomial time many-one reduction.

3. Basic planar counting problems. In this section, we prove that the problem PL-3SAT has a parsimonious crossover box. Specifically, we show that the crossover box in [15] is parsimonious. One immediate corollary is that the problem #PL-3SAT is #P-hard. We also prove that the problem 3SAT is planarity-preserving and parsimoniously reducible to each of the basic SAT problems listed in Table 1.1.

DEFINITION 3.1. *A* crossover box *for a satisfiability problem Π is a formula f_c with four distinguished variables a, a_1, b, b_1, which can be laid out on the plane with the distinguished variables on the outer face, such that*

(1) *the* old variables *a and b are opposite to the corresponding* new variables *a_1 and b_1,*

(2) *each assignment to a and b can be extended to a satisfying assignment of f_c,*

(3) *for any satisfying assignment of f_c, $a \equiv a_1$ and $b \equiv b_1$.*

The crossover box is parsimonious *if and only if for each assignment to the old variables, there is exactly one extension of this assignment to the variables in the crossover box such that f_c is satisfied.*

THEOREM 3.2. *The problem* 3SAT *has a parsimonious planar crossover box. Hence,* 3SAT *is parsimoniously reducible to* PL-3SAT *and* #PL-3SAT *is* #P*-complete.*

Proof. The crossover box described here is the same as the one given in [15]. Here we prove that the crossover box also preserves the number of solutions. For expository purposes, we describe the crossover box in two steps. The first step is to consider the

following formula:

$$f_c = (\overline{a_2} + \overline{b_2} + \alpha) \wedge (a_2 + \overline{\alpha}) \wedge (b_2 + \overline{\alpha}) \bigwedge$$

$$(\overline{a_2} + b_1 + \beta) \wedge (a_2 + \overline{\beta}) \wedge (\overline{b_1} + \overline{\beta}) \bigwedge$$

$$(a_1 + b_1 + \gamma) \wedge (\overline{a_1} + \overline{\gamma}) \wedge (\overline{b_1} + \overline{\gamma}) \bigwedge$$

$$(a_1 + \overline{b_2} + \delta) \wedge (\overline{a_1} + \overline{\delta}) \wedge (b_2 + \overline{\delta}) \bigwedge$$

$$(\alpha + \beta + \gamma + \delta) \bigwedge$$

$$(\overline{\alpha} + \overline{\beta}) \wedge (\overline{\beta} + \overline{\gamma}) \wedge (\overline{\gamma} + \overline{\delta}) \wedge (\overline{\delta} + \overline{\alpha}) \bigwedge$$

$$(a_2 + \overline{a}) \wedge (a + \overline{a_2}) \wedge (b_2 + \overline{b}) \wedge (b + \overline{b_2}).$$

Following [15], we give an intuitive explanation of formula f_c. Clauses 1, 2, and 3 imply that $(\alpha \leftrightarrow (a_2 \wedge b_2))$; clauses 4, 5, and 6 imply $(\beta \leftrightarrow (a_2 \wedge \overline{b_1}))$; clauses 7, 8, and 9 imply $(\gamma \leftrightarrow (\overline{a_1} \wedge \overline{b_1}))$; clauses 10, 11, and 12 imply $(\delta \leftrightarrow (\overline{a_1} \wedge b_2))$. Clause 13 (the four literal clause) implies that at least one of α, β, γ, or δ is true. Clauses 14, 15, 16, 17 imply that $(\alpha+\gamma) \rightarrow (\overline{\beta} \wedge \overline{\delta})$ and $(\beta+\delta) \rightarrow (\overline{\alpha} \wedge \overline{\gamma})$. Finally, clauses 18 and 19 imply $(a \leftrightarrow a_2)$ and $(b \leftrightarrow b_2)$. It can now be verified that the formula f_c implies $(a_1 \leftrightarrow a)$ and $(b_1 \leftrightarrow b)$. For example, consider an assignment $\mathbf{v}$ such that $\mathbf{v}[a_1] = \mathbf{v}[b_1] = 0$. Then f_c implies that $\mathbf{v}[\beta] = \mathbf{v}[\delta] = \mathbf{v}[\alpha] = \mathbf{v}[a_2] = \mathbf{v}[b_2] = 0$ and $\mathbf{v}[\gamma] = 1$. We leave it to the reader to verify the other three cases. Thus, in any satisfying assignment to f_c, the new variables $a_1, a_2, b_1, b_2, \alpha, \beta, \gamma$, and δ are *functionally* dependent on a and b. In other words, given an assignment to the variable a and b there is a unique way to extend this assignment so as to satisfy all the clauses in f_c. Thus f_c is a parsimonious crossover box. Even though the formula itself is a parsimonious planar crossover box, it is unsuitable for a reduction to Pl-3Sat because it has one clause with four literals, namely, $(\alpha + \beta + \gamma + \delta)$. The second step is to obtain the formula f_c' by replacing this clause with the formula $(\alpha + \delta + \xi) \wedge (\overline{\xi} + \beta + \gamma)$. The planarity of the formula f_c' is demonstrated in Figure 3.1. This step also preserves numbers of satisfying assignments as demonstrated by the following claim.

Claim 3.3. (1) *Exactly one of $\alpha, \beta, \gamma, \delta$ is true in any satisfying assignment to f_c.* (2) *ξ is functionally dependent on α, β, γ, and δ. Thus a satisfying assignment for f_c can be extended in a unique way to a satisfying assignment to the formula f_c'.*

Proof. We prove the claim for the case when α is true. The other three cases are similar. As already discussed, clauses 14, 15, 16, 17 imply that $(\alpha+\gamma) \rightarrow (\overline{\beta} \wedge \overline{\delta})$ and $(\beta + \delta) \rightarrow (\overline{\alpha} \wedge \overline{\gamma})$. Consider a satisfying assignment $\mathbf{v}$ such that $\mathbf{v}[\alpha] = 1$. Then the above discussion implies that $\mathbf{v}[\beta] = \mathbf{v}[\delta] = 0$. Now, since $(\beta \leftrightarrow (a_2 \wedge \overline{b_1}))$ and $\mathbf{v}[\beta] = 0$, we have $\mathbf{v}[a_2] = 1$ and $\mathbf{v}[b_1] = 1$. Since $(\gamma \leftrightarrow (\overline{a_1} \wedge \overline{b_1}))$ and $\mathbf{v}[b_1] = 1$ it implies that $\mathbf{v}[\gamma] = 0$. This forces $\mathbf{v}[\xi] = 0$. □

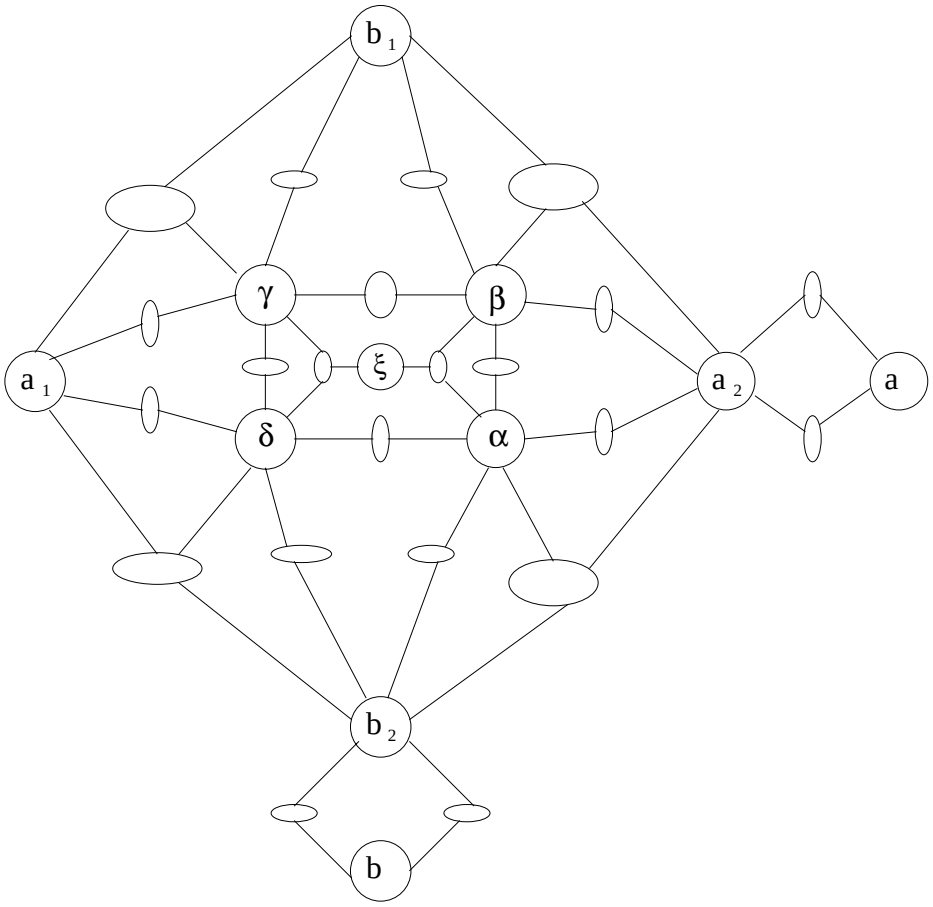


FIG. 3.1. *The parsimonious planar crossover box for* 3SAT. *The clauses are denoted by ellipses and the variables are denoted by circles.*

Thus, the satisfying assignments to f_c' satisfy f_c. It can now be seen that in a satisfying assignment to the variables of f_c', the values of a_1, a_2, b_1, b_2, α, β, γ, δ, and ξ are all functionally dependent on a and b. Therefore, f_c' is a parsimonious crossover box.

We can now describe the reduction given in [15] from 3SAT to PL-3SAT. For any 3CNF formula, lay the formula in the plane, possibly with certain edge pairs crossing over at "crossover points." This layout is a planar graph with vertex set consisting of the variables, clauses, and crossover points. In this layout, we add a new variable node on the edge between two crossover points or between a crossover point and a clause node, as shown in Figure 3.2. The resulting graph is bipartite where the first set of nodes consists of the variable nodes and the second set consists of the clause nodes and the crossover points. Each edge is between a variable and a crossover point, or between a variable and a clause. Also, each crossover point has four distinct variables as neighbors. We now replace each crossover point with the crossover box in Figure 3.1, where a_2, b_2, α, β, γ, δ, and ξ are given distinct names in each replacement. Here a, b, a_1, and b_1 are identified with the neighbors of the crossover point in cyclic order in the layout. The new layout is planar and the

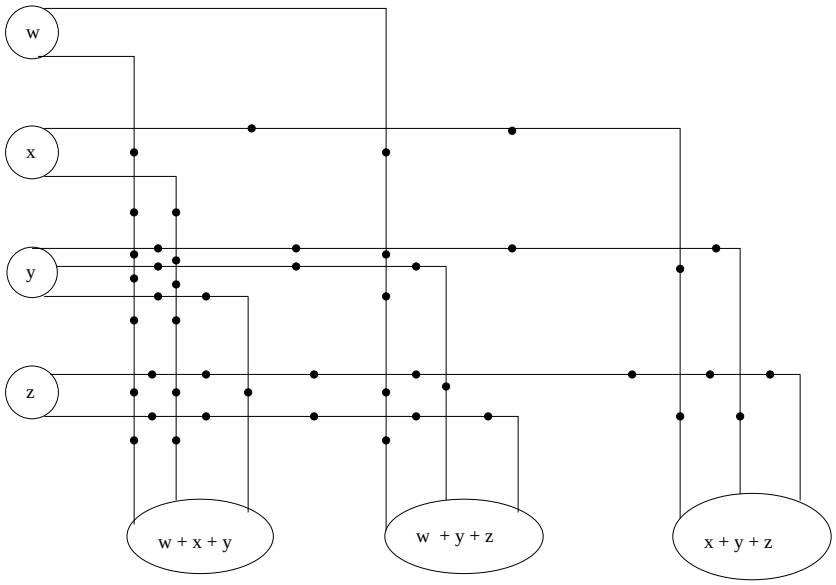


FIG. 3.2. *Figure showing how new variables are introduced in the bipartite graph for* 3SAT *to obtain an instance of* PL-3SAT. *In the example, the instance of* 3SAT *is given by* $F = (w + x + y) \wedge (w + y + z) \wedge (x + y + z)$. *The circles denote original variables, ellipses denote the original clauses, and the black dots denote new variables added. Each crossover of edges is replaced by a crossover box shown in Figure* 3.1.

new formula has the same number of solutions as the original, since in any satisfying assignment the new variables are functionally dependent on the old. □

Next, we strengthen Theorem 3.2 by showing that counting the number of satisfying assignments of a planar 3CNF formula is #P-complete, even when the input consists of a planar 3CNF formula F and an assignment $\mathbf{v}[F]$ to the variables of F which satisfies F.

THEOREM 3.4. *Given an instance* F *of* PL-3SAT *and an assignment* $\mathbf{v}$ *to the variables of* F *such that* $\mathbf{v}[F] = 1$, *the problem of counting the number of satisfying assignments of* F *is* #P*-complete.*

Proof. Step 1. Given an arbitrary planar 3CNF formula $f(x_1, x_2, \ldots, x_n)$, we first construct a new formula $f_1(x_1, x_2, \ldots, x_n, x_{n+1})$, where $f_1 = (f \wedge x_{n+1}) \bigvee [\overline{x_1} \wedge \overline{x_2} \wedge \cdots \wedge \overline{x_n} \wedge \overline{x_{n+1}}]$.

Obviously, an assignment $\mathbf{v}$, such that $\mathbf{v}[x_1] = \mathbf{v}[x_2] = \cdots = \mathbf{v}[x_n] = \mathbf{v}[x_{n+1}] = 0$, satisfies f_1. Hence f_1 is always satisfiable. Also, the number of satisfying assignments of f_1 is one more than the number of satisfying assignments of f. Therefore, knowing the number of satisfying assignments of f_1 tells us the number of satisfying assignments of f.

Step 2. Convert $f_1(x_1, x_2, \ldots, x_n, x_{n+1})$ into an equivalent 3CNF formula $f_2(x_1, x_2, \ldots, x_n, x_{n+1}, x_{n+2}, \ldots, x_{n+p})$, where $x_{n+2}, \ldots, x_{n+p}$ are new variables. This is done in the standard way as follows. Obtain a parse tree of f_1. For each non-leaf node in the parse tree introduce new variables $y_1, y_2, \ldots, y_m$, where y_m is the variable corresponding to the root of the parse tree. Each node of the parse tree corresponds to an operator applied to one or two inputs. Let the children of a non-leaf node i be j and k.

Case 1. If the operator at node i is an AND operator, construct a new 3CNF formula equivalent to the formula $y_i \equiv (y_j \wedge y_k)$.

Case 2. If the operator is an OR operator, construct a 3CNF formula equivalent to the formula $y_i \equiv (y_j + y_k)$.

Case 3. If the operator is a NOT operator, construct a 3CNF formula equivalent to the formula $y_i \equiv \overline{y_j}$.

The final 3CNF formula f_2 is a conjunction of all the 3CNF formulas along with y_m. Now it is easy to see that f_2 is satisfiable if and only if f_1 is satisfiable and the reduction is parsimonious. (The new variables are functionally dependent on the old.)

Step 3. Next, lay out f_2 on a plane and replace each crossover in the layout by the crossover box described in Theorem 3.2. Let f_3 be the resulting planar 3CNF formula. By Theorem 3.2, the reduction from f_2 to f_3 is parsimonious. Thus the number of satisfying assignments of f_3 is one more than the number of satisfying assignments of f. The theorem now follows. □

Next we extend this result to prove that counting the number of satisfying assignments of a 1-*valid* planar 3CNF formula is #P-hard.

THEOREM 3.5. *The problem* #1-VALID PL-3SAT *is* #P*-complete.*

Proof. Given an arbitrary PL-3CNF formula $f(x_1, x_2, \ldots, x_n)$ and a satisfying assignment $\mathbf{v}$ such that $\mathbf{v}[\mathrm{f}] = 1$, we construct a new formula $f_1(x_1, x_2, \ldots, x_n, y_1, \ldots, y_p)$, where $y_1, \ldots, y_p$ are new variables. The formula f_1 is constructed as follows. Let $x_{l_1}, x_{l_2}, \ldots, x_{l_p}$ be the variables in f such that $\mathbf{v}[x_{l_1}] = \mathbf{v}[x_{l_2}] = \cdots = \mathbf{v}[x_{l_p}] = 0$. Then replace $\overline{x_{l_i}}$ by y_i and x_{l_i} by $\overline{y_i}$, $1 \leq i \leq p$. Obviously, the formula f_1 is 1-valid and the reduction is parsimonious. □

Karp and Luby [14] presented randomized fully polynomial time approximation schemes for several #P-complete problems, including #DNF. Since then, substantial research has been done in the area of approximation algorithms for various counting problems. Saluja, Subramanium, and Thakur [22] give a logical characterization of the counting problems that have a polynomial time random approximation scheme. Our #P-hardness of #PL-3SAT and other problems immediately raise the question of approximating the optimal values of these counting problems. The parsimonious reduction from 3SAT to PL-3SAT implies that, given a deterministic polynomial time algorithm A to approximately count the number of satisfying assignments of a planar 3CNF formula, we can construct a deterministic polynomial time algorithm A' with the same performance guarantee to approximately count the number of satisfying assignments of an arbitrary 3CNF formula.

Intuitively Theorem 3.2 and the above observation mean that counting the number of satisfying assignments of a planar 3CNF formula is *as hard as* counting the number of satisfying assignments of an arbitrary 3CNF formula with respect to polynomial time reducibility. We remark that the result holds even for 1-VALID PL-3CNF formulas.

Next, we prove the #P-hardness of other basic satisfiability problems. First we prove two lemmas.

LEMMA 3.6. *Let F be the planar monotone formula $(c + d + e) \wedge (c + e + f) \wedge (d + e + f)$. Then there is a unique satisfying assignment $\mathbf{v}$ to the variables of F such that each clause has exactly one true literal, namely, $\mathbf{v}[c] = \mathbf{v}[d] = \mathbf{v}[f] = 0$; and $\mathbf{v}[e] = 1$.*

Proof. The proof is by inspection. □

LEMMA 3.7. *The following* EX3CNF *formula is planar and has exactly one satisfying assignment, namely, the assignment $\mathbf{v}$ defined by $\mathbf{v}[x_i] = 0 (1 \leq i \leq 9)$:*

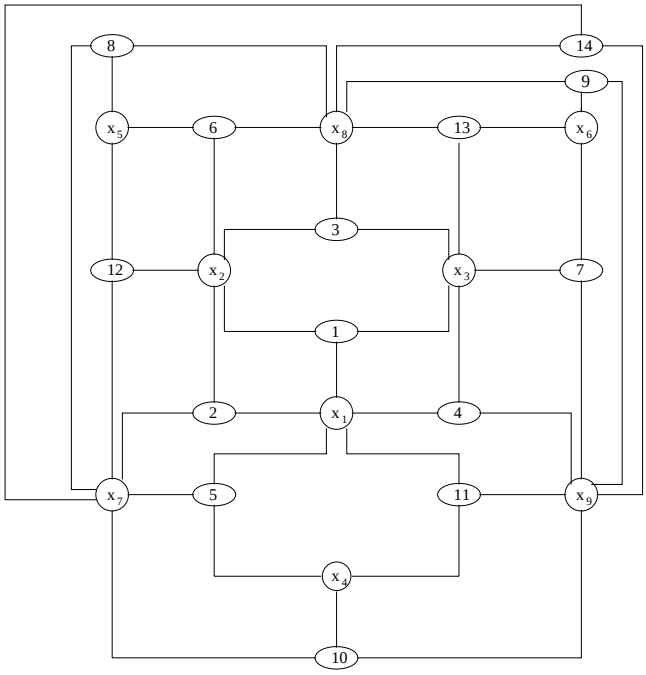


FIG. 3.3. PL-EX3SAT *formula as described in Lemma* 3.7. *Ellipses denote the clauses in* G. *The clauses are numbered in the order in which they appear in Lemma* 3.7.

$G(x_1, \ldots, x_9) = (\overline{x}_1 + \overline{x}_2 + \overline{x}_3) \wedge (\overline{x}_1 + x_2 + x_7) \wedge (\overline{x}_2 + x_3 + x_8) \wedge (\overline{x}_3 + x_1 + x_9) \wedge (\overline{x}_4 + x_1 + x_7) \wedge (\overline{x}_5 + x_2 + x_8) \wedge (\overline{x}_6 + x_3 + x_9) \wedge (\overline{x}_7 + x_5 + x_8) \wedge (\overline{x}_8 + x_6 + x_9) \wedge (\overline{x}_9 + x_4 + x_7) \wedge (\overline{x}_1 + \overline{x}_4 + \overline{x}_9) \wedge (\overline{x}_2 + \overline{x}_5 + \overline{x}_7) \wedge (\overline{x}_3 + \overline{x}_6 + \overline{x}_8) \wedge (\overline{x}_7 + \overline{x}_8 + \overline{x}_9)$.

Proof. Planarity is demonstrated by Figure 3.3. The given assignment $\mathbf{v}$ satisfies G since each clause of G contains a negated literal.

Suppose G is true. Let $\mathbf{u}$ be any truth assignment to the variables in G. Then, $x_7 \rightarrow x_8$ by clauses 6, 8, and 12; $x_8 \rightarrow x_9$ by clauses 7, 9, and 13; $x_9 \rightarrow x_7$ by clauses 5, 10, and 11. Therefore, $(x_7 + x_8 + x_9) \rightarrow (x_7 \wedge x_8 \wedge x_9)$. Hence by clause 14, $\mathbf{u}[x_7] = \mathbf{u}[x_8] = \mathbf{u}[x_9] = 0$. Given this, $x_1 \rightarrow x_2$ by clause 2; $x_2 \rightarrow x_3$ by clause 3; and $x_3 \rightarrow x_1$ by clause 4. Therefore, $(x_1 + x_2 + x_3) \rightarrow (x_1 \wedge x_2 \wedge x_3)$. Hence by clause 1, $\mathbf{u}[x_1] = \mathbf{u}[x_2] = \mathbf{u}[x_3] = 0$. But this implies that $\mathbf{u}[x_4] = \mathbf{u}[x_5] = \mathbf{u}[x_6] = 0$ by clauses 5, 6, and 7. Hence in any satisfying assignment $\mathbf{u}$ of G, $\mathbf{u}[x_i] = 0$ $(1 \leq i \leq 9)$. □

Next we give planarity-preserving parsimonious reductions from 3SAT to the basic SAT problems listed in Table 1.1. Without loss of generality, we assume that the given instance of the CNF formula does not have any single literal clause.

THEOREM 3.8. *There exist planarity-preserving parsimonious reductions from* 3SAT *to each of the following problems:* EX3SAT, 1-3SAT, 1-EX3SAT, 1-EX3MONOSAT *and* X3C.

Proof. 3SAT $\rightarrow$ EX3SAT: Let f be a 3CNF formula with clauses $c_j (1 \leq j \leq k)$. For $1 \leq j \leq k$, let c'_j be c_j, if c_j is a three-literal clause. If $c_j = (l_{j1} + l_{j2})$, let c'_j be $(l_{j1} + l_{j2} + x_9^j) \wedge G(x_1^j, \ldots, x_9^j)$.

G is defined as in Lemma 3.7 and x_i^j $(1 \leq i \leq 9)$ are distinct new variables. Let

$$g = \bigwedge_{j=1}^{k} c'_j.$$

Then by Lemma 3.7, Figure 3.3, and direct inspection of the definitions of the formulas c'_j, the reduction mapping f into g is seen to be a planarity-preserving and parsimonious reduction of the problem 3SAT to the problem EX3SAT.

3SAT $\rightarrow$ 1-EX3SAT[2]: Let f be a 3CNF formula with clauses $c_j (1 \leq j \leq m)$.

(1) For each three-literal clause $c_j = (z_p + z_q + z_r)$ of f, let $c'_j = (z_p + u^j + v^j) \wedge (\overline{z_q} + u^j + w^j) \wedge (v^j + w^j + t^j) \wedge (\overline{z_r} + v^j + x^j)$, where u^j, v^j, w^j, t^j, and x^j are distinct new variables local to c'_j.

(2) For each two-literal clause $c_j = (z_p + z_q)$ of f, let $c'_j = (z_p + u^j + v^j) \wedge (\overline{z_q} + u^j + w^j) \wedge (v^j + w^j + t^j) \wedge (\overline{a^j} + v^j + x^j) \wedge (a^j + d^j + e^j) \wedge (a^j + e^j + f^j) \wedge (d^j + e^j + f^j)$, where u^j, v^j, w^j, t^j, x^j, a^j, d^j, and e^j are all new variables local to c'_k. Let

$$f' = \bigwedge_{j=1}^{m} c'_j.$$

To see the planarity of this reduction, see Figure 3.4. We claim that f' is exactly-one satisfiable *if and only if* the original formula f is satisfiable. Moreover, the reduction is planarity-preserving and parsimonious. To prove that the reduction is parsimonious, it suffices to show the following two claims.

CLAIM 3.9. *No assignment of truth values to the variables of clause c_j $(1 \leq j \leq m)$ of f which does not satisfy c_j can be extended to an assignment of truth values to the variables of the formula c'_j that exactly-one satisfies c'_j.*

Proof. Let $c_j = (z_p + z_q + z_r)$ and $\mathbf{u}$ be an assignment to the variables such that $\mathbf{u}[z_p] = \mathbf{u}[z_q] = \mathbf{u}[z_r] = 0$. Let $\mathbf{v}$ be an exactly-one satisfying assignment to the variables of c'_j such that $\mathbf{v}[z_p] = \mathbf{v}[z_q] = \mathbf{v}[z_r] = 0$ (i.e., $\mathbf{v}$ is an extension of $\mathbf{u}$). Then, the clauses $(\overline{z_q} + u^j + w^j)$ and $(\overline{z_r} + v^j + x^j)$ imply that $\mathbf{v}[u^j] = \mathbf{v}[w^j] = \mathbf{v}[v^j] = \mathbf{v}[x^j] = 0$. It follows that $\mathbf{v}$ does not exactly-one satisfy the clause $(z_p + u^j + w^j)$ of c'_j.

Let $c_j = (z_p + z_q)$ and $\mathbf{u}$ be an assignment to the variables such that $\mathbf{u}[z_p] = \mathbf{u}[z_q] = 0$. Let $\mathbf{v}$ be an exactly-one satisfying assignment to the variables of c'_j such that $\mathbf{v}[z_p] = \mathbf{v}[z_q] = 0$. Then, the clauses $(z_p + u^j + v^j)$ and $(\overline{z_q} + u^j + w^j)$ imply that $\mathbf{v}[u^j] = \mathbf{v}[w^j] = 0$ and $\mathbf{v}\ [v^j] = 1$. But given this, the clause $(\overline{a^j} + v^j + x^j)$ implies that $\mathbf{v}[a^j] = 1$. Lemma 3.6 now implies that $\mathbf{v}$ does not exactly-one satisfy c'_j. □

CLAIM 3.10. *For each satisfying assignment to the variables of the clause c_j $(1 \leq j \leq m)$ of f, there is exactly one way the assignment can be extended to the variables of the formula c'_j so as to exactly-one satisfy c'_j.*

Proof. When $c_j = (z_p + z_q + z_r)$, we need to verify that the only exactly-one satisfying assignments c'_j are the following:

1. $z_p = 1,\ z_q = 0,\ z_r = 0,\ u^j = 0, v^j = 0,\ w^j = 0,\ t^j = 1,\ x^j = 0;$
2. $z_p = 0,\ z_q = 1,\ z_r = 0,\ u^j = 1,\ v^j = 0,\ w^j = 0,\ t^j = 1,\ x^j = 0;$
3. $z_p = 0,\ z_q = 0,\ z_r = 1,\ u^j = 0,\ v^j = 1,\ w^j = 0,\ t^j = 0,\ x^j = 0;$

[2] Although they claim to have a parsimonious reduction, the reduction actually given in [3] is *not* parsimonious.

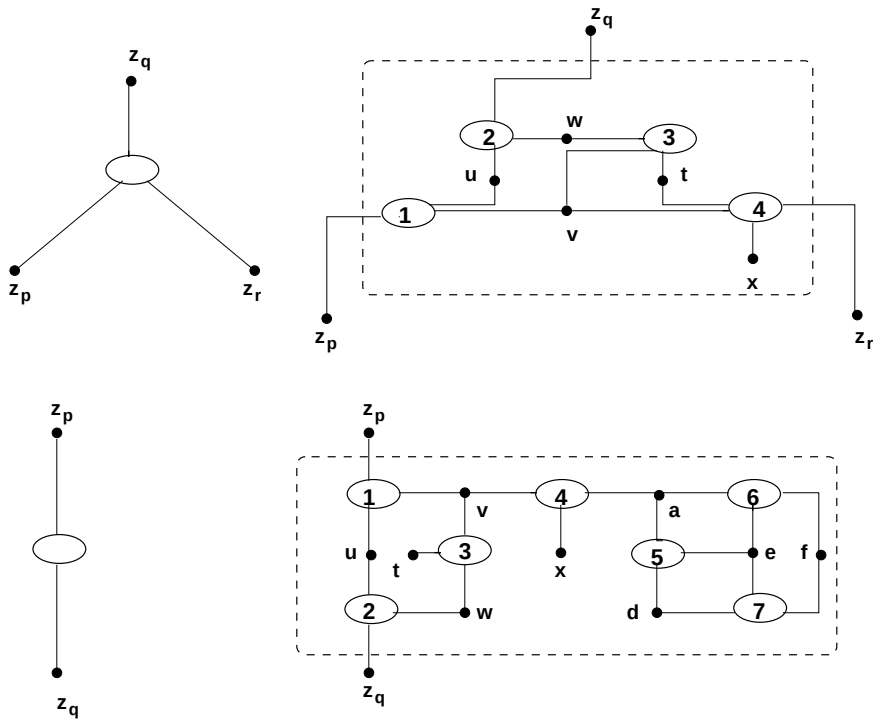


FIG. 3.4. *Figure illustrating the reduction from* 3SAT *to* 1-EX3SAT. (a) *shows how to transform a three-literal clause.* (b) *shows how to transform a two-literal clause. The clauses are numbered in the order in which they appear in the reduction outlined in the proof of Theorem* 3.8. *Note that the reduction is a local replacement–type reduction and hence preserves planarity.*

4. $z_p = 1$, $z_q = 1$, $z_r = 0$, $u^j = 0$, $v^j = 0$, $w^j = 1$, $t^j = 0$, $x^j = 0$;
5. $z_p = 1$, $z_q = 0$, $z_r = 1$, $u^j = 0$, $v^j = 0$, $w^j = 0$, $t^j = 1$, $x^j = 1$;
6. $z_p = 0$, $z_q = 1$, $z_r = 1$, $u^j = 1$, $v^j = 0$, $w^j = 0$, $t^j = 1$, $x^j = 1$;

and

7. $z_p = 1$, $z_q = 1$, $z_r = 1$, $u^j = 0$, $v^j = 0$, $w^j = 1$, $t^j = 0$, $x^j = 1$.

When $c_j = (z_p + z_q)$, we need to verify that the only exactly-one assignments of c'_j are the following:

1. $z_p = 1$, $z_q = 0$, $u^j = 0$, $v^j = 0$, $w^j = 0$, $t^j = 1$, $x^j = 0$, $c^j = 0$, $d^j = 0$, $e^j = 1$, $f^j = 0$;
2. $z_p = 0$, $z_q = 1$, $u^j = 1$, $v^j = 0$, $w^j = 0$, $t^j = 1$, $x^j = 0$, $c^j = 0$, $d^j = 0$, $e^j = 1$, $f^j = 0$; and
3. $z_p = 1$, $z_q = 1$, $u^j = 0$, $v^j = 0$, $w^j = 1$, $t^j = 0$, $x^j = 0$, $c^j = 0$, $d^j = 0$, $e^j = 1$, $f^j = 0$. □

1-EX3SAT → 1-EX3MONOSAT: Let f be an instance of 1-EX3SAT. Let

$$f = \bigwedge_{j=1}^{m} c_j.$$

For each c_j, construct c'_j as follows. Replace each negated literal of the form $\overline{z_p}$ appearing in the clause c_j by a distinct new variable y_p^j in c_j, then add the clauses $(z_p + y_p^j + a_p^j) \wedge (a_p^j + d_p^j + e_p^j) \wedge (a_p^j + f_p^j + e_p^j) \wedge (d_p^j + f_p^j + e_p^j)$. Note that for each negated

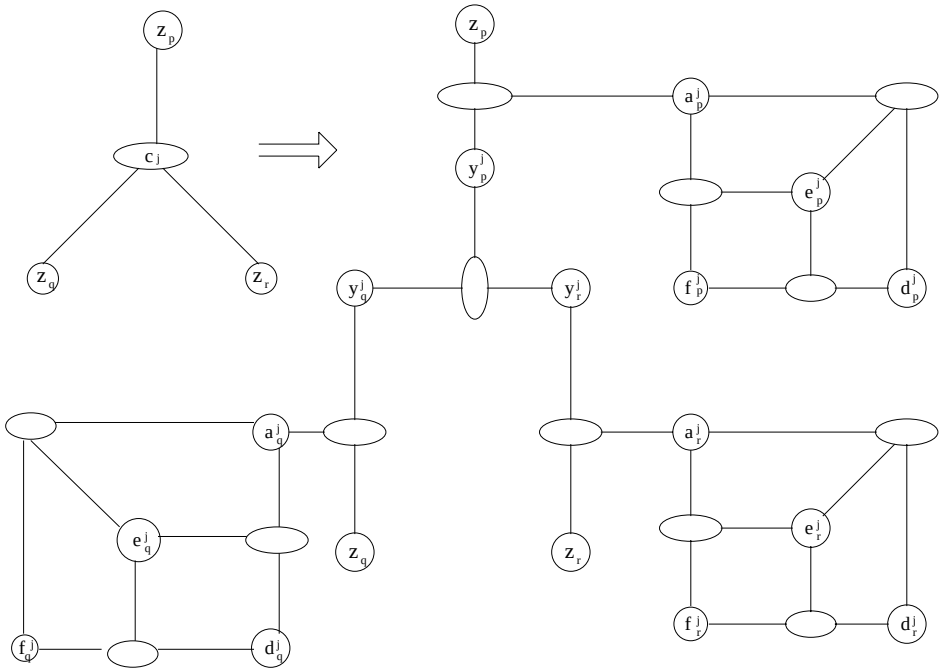


FIG. 3.5. *Figure illustrating the reduction from* 1-EX3SAT *to* 1-EX3MONOSAT. *The figure illustrates the construction for a three-literal clause c_j which contains all negated literals.*

literal, we introduce new copies of the auxiliary variables $a_p^j, \ldots, f_p^j$. See Figure 3.5 for an example.

Let

$$f' = \bigwedge_{j=1}^{m} c'_j.$$

Then f' is an instance of 1-EX3MONOSAT obtained from f. The result follows from Lemma 3.6 and the fact that for all variables x and y, $(x+y)$ is exactly-one satisfiable *if and only if* $x = \overline{y}$.

1-EX3MONOSAT $\rightarrow$ X3C: Although Dyer and Frieze [3] do not observe this, the reduction given in their paper [3] from 1-EX3SAT to X3C is actually parsimonious. The reduction presented here is essentially the same as that given in [3], except that we start from an instance of 1-EX3MONOSAT. Thus in our reduction, we do not have to take care of negated literals. We now describe the reduction. Each variable is represented by a cycle of 3 element sets. If the variable occurs r times in the 1-EX3SAT instance, then there are $2r$ sets, with each successive pair of sets sharing an element. This cycle is augmented with r additional sets and $2r$ elements by adding a 3-set to one of the external elements in each pair. The 3 elements now corresponding to an appearance of v_j will be called a connector. The variable v_j is set to true if and only if all three connector elements are covered by the cycle when v_j appears in the corresponding clause. Figure 3.6 illustrates the variable component. Next, consider each clause c_i. Each c_i is represented by a configuration shown in Figure 3.7. This has 12 elements and 9 sets. Of the 12 elements, 3 are internal and the rest are grouped in groups of 3. Each group of 3 elements is called a terminal of c_i. Finally, we connect a

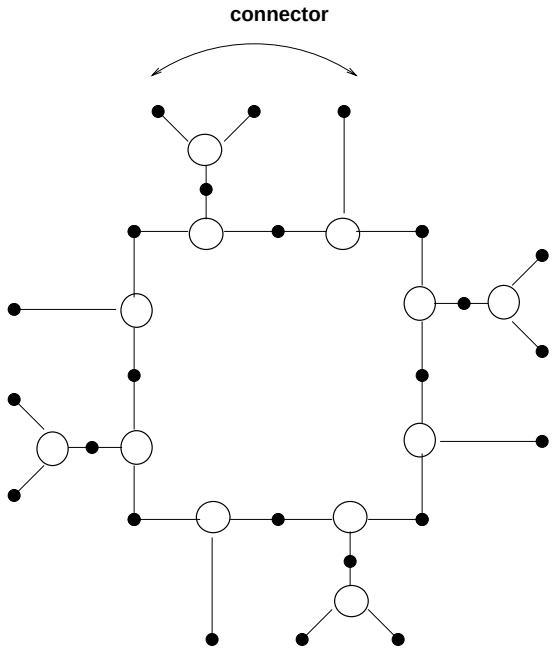


FIG. 3.6. *Variable configuration for reduction to* X3C. *The black dots represent element nodes, while the ellipses denote the triples.*

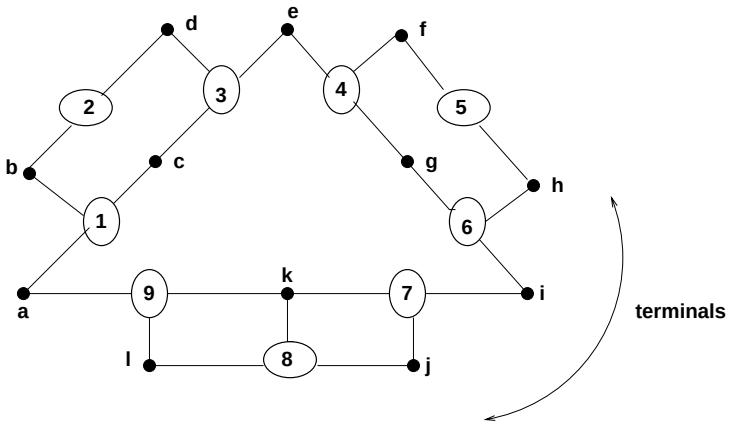


FIG. 3.7. *Figure illustrating the clause configuration. The sets* $\{l, a, b\}, \{d, c, f\}, \{j, i, h\}$ *represent the three terminals. The vertices labeled* c, g, k *are the internal elements.*

clause component to the variable component as follows. For each $v_j \in c_i$ we identify three distinct connector elements with one of the terminals in c_i. The construction is depicted in Figure 3.8. Let G denote the graph obtained as a result of the construction. Planarity of G follows by the fact that each component replacing a variable and a clause is planar and the components are joined in a planarity-preserving way. We first prove that there is an exact cover of c_i configuration if and only if exactly one terminal is covered externally, when we restrict the covering such that either none or all 3 of the elements in each terminal are covered externally. But the configuration has the

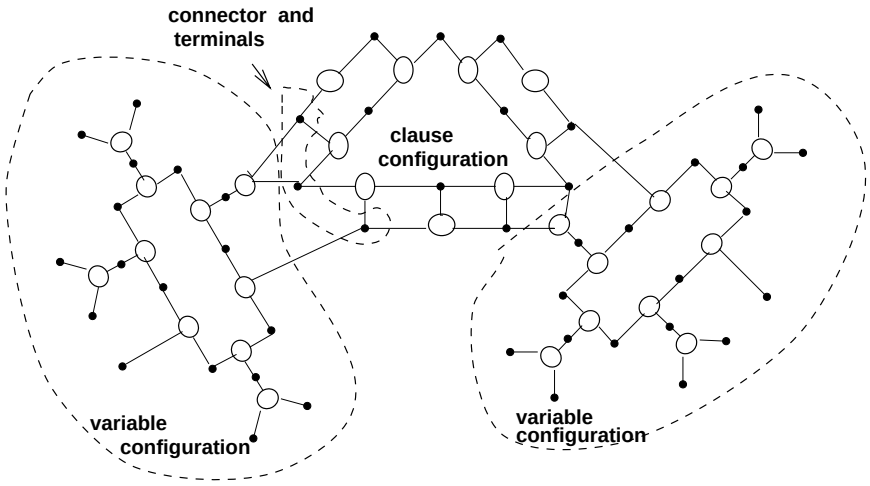


FIG. 3.8. *Figure illustrating the way the clause and variable configurations are attached.*

property that each of the 3 internal elements appear in 3 of the 9 sets and no 2 appear in the same set. It follows that if this configuration forms part of an exact cover by 3-sets, then exactly 3 of the sets must be used, hence 9 of the 12 elements will be covered internally. Moreover, this can only be done so that exactly 1 of the terminals will be left uncovered. This uncovered terminal is covered by sets in the variable configuration and amounts to setting the literal true. It can then be argued that the exact cover by 3-sets has a solution if and only if the corresponding 1-EX3MONOSAT instance is satisfiable. It is easily verified that the reduction is parsimonious. This is because the clause configuration forces precisely 1 literal to be set to true and the other 2 literals to be false. Moreover, for each satisfying assignment of 1-EX3MONOSAT, there is exactly 1 way the sets can be chosen so as to have an exact cover. Hence, the reduction is parsimonious. □

COROLLARY 3.11. *The problems* PL-EX3SAT, PL-1-EX3SAT, *and* PL-1-EX3MONOSAT *are* NP*-complete. The problems* #PL-EX3SAT, #PL-1-3SAT, #PL-1-EX3SAT, #PL-1-EX3MONOSAT, *and* #PL-X3C *are* #P*-complete.*

4. Planar graph problems.

4.1. Overview of our proofs. In this section we give parsimonious/weakly parsimonious and planarity-preserving reductions from PL-1-EX3MONOSAT to various graph problems. The problems considered here are MINIMUM VERTEX COVER, MINIMUM DOMINATING SET, CLIQUE COVER, FEEDBACK VERTEX SET, PARTITION INTO CLAWS, PARTITION IN TRIANGLES, and BIPARTITE DOMINATING SET. Previously, reductions showing that these problems were NP-hard frequently did not preserve the number of solutions. Central to the proofs is the reduction (called $RED1$) from 1-EX3MONOSAT to 3SAT with the property that every formula is mapped to a formula in which all satisfying assignments satisfy exactly one literal in each three-literal clause. This in turn enables us to obtain (weakly) parsimonious reductions from EX1-3MONOSAT to the problems considered here.

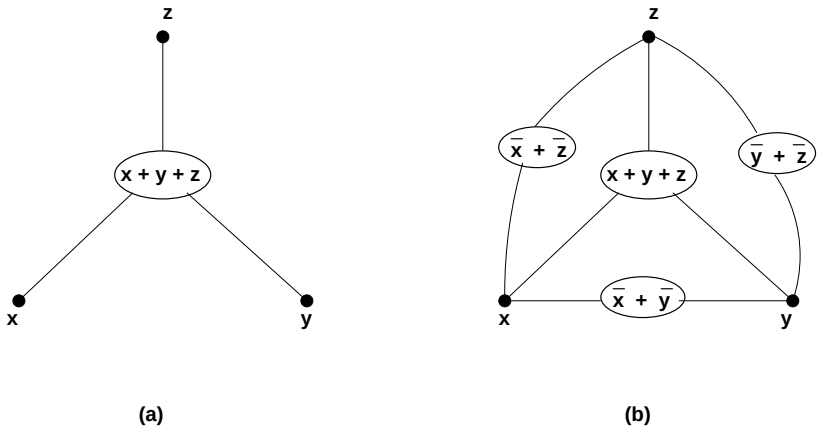


FIG. 4.1. *Figure illustrating the reduction RED*1 *discussed in section* 4.2. *Observe that the reduction is planarity-preserving.*

4.2. Reduction RED1. Let *RED*1 be a mapping from an instance

$$f = \bigwedge_{j=1}^{m} c_j$$

of 1-EX3MONOSAT to an instance

$$f' = \bigwedge_{j=1}^{m} c'_j$$

of 3SAT, where for $c_j = (x + y + z)$, $c'_j = (x + y + z) \wedge (\overline{x} + \overline{y}) \wedge (\overline{x} + \overline{z}) \wedge (\overline{z} + \overline{y})$.

LEMMA 4.1. *The formula* f' *has the following properties.*

(1) *The satisfying assignments of* f' *are* exactly *the exactly-one satisfying assignments of* f.

(2) *In any satisfying assignment of* f', *all but one clause in* c'_j *is exactly-one satisfiable.*

(3) *Each variable in the formula* f' *occurs at least twice negated and at least once unnegated.*

(4) *RED*1 *is planarity-preserving. (See Figure* 4.1.*)*

(5) *RED*1 *is parsimonious.*

Proof. The proof is by inspection. □

We call each of the c'_j a *clause group.* Observe that each c'_j has four clauses; one is a three-literal clause and the others are two-literal clauses.

4.3. Weakly parsimonious reductions and basic graph problems.

THEOREM 4.2. *There exists a planarity-preserving and weakly parsimonious reduction from* 1-EX3MONOSAT *to each of the following problems:* (1) MINIMUM VERTEX COVER, (2) MINIMUM DOMINATING SET, (3) MINIMUM FEEDBACK VERTEX SET, *and* (4) SUBGRAPH ISOMORPHISM.

Proof. (1) MINIMUM VERTEX COVER: The reduction is from 1-EX3MONOSAT and is similar to the one given in [5] for proving NP-hardness of MINIMUM VERTEX COVER. Let f be a 1-EX3MONOSAT formula. Apply *RED*1 to f to obtain f'. Next,

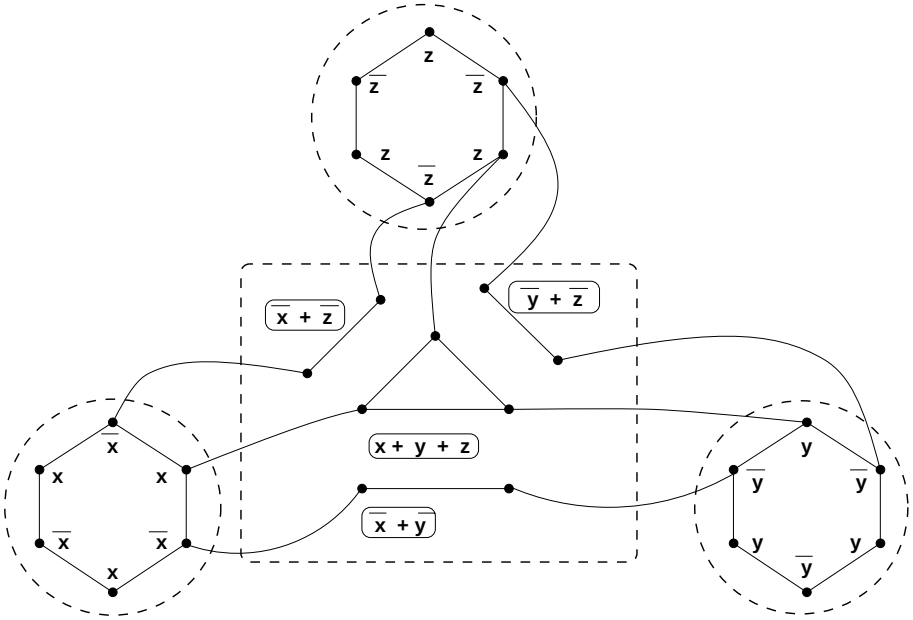


FIG. 4.2. *Figure illustrating the reduction from* 1-EX3MONOSAT *to* VERTEX COVER *and the transformation of a clause group* $(x + y + z) \wedge (\overline{x} + \overline{y}) \wedge (\overline{x} + \overline{z}) \wedge (\overline{z} + \overline{y})$. *The dotted enclosures depict how to locally transform the clauses as well as the variables so as to preserve planarity of the resulting graph.*

starting from f', construct an instance $G(V, E)$ of the vertex cover problem as shown in Figure 4.2 as follows.

1. Consider a clause group $c'_j = (x + y + z) \wedge (\overline{x} + \overline{y}) \wedge (\overline{x} + \overline{z}) \wedge (\overline{z} + \overline{y})$. Corresponding to the clause $(x + y + z)$, construct a triangle with vertices $\{x, y, z\}$ and edges $\{(x, y), (x, z), (y, z)\}$. Corresponding to a clause of the form $(\overline{x} + \overline{y})$ add the edge $\{(x, y)\}$. Call this the clause graph.

2. For each variable x that appears i times we construct a simple cycle with $2i$ vertices. Let the odd-numbered variables represent the negated occurrences and the even-numbered variables represent the unnegated occurrences. Call this the variable graph.

3. Join the vertices of the clause graph to the vertices of the variable graph as follows. Consider a clause group c'_j. Corresponding to a clause, join the triangle vertices x, y, and z to the corresponding unnegated occurrences of x, y, and z in the cycles. Corresponding to a clause of the form $(\overline{x} + \overline{y})$, join the two vertices x, y to the negated occurrence of the variables x and y, respectively. Repeat the procedure for each clause group.

Now set $K = 1/2 \sum C_i + 2m + 3m$, where C_i is the length of the cycle of the variable i and m is the number of clause groups in f'. The reduction is illustrated in Figure 4.2.

CLAIM 4.3. (1) *The formula f' is satisfiable if and only if the graph G has a vertex cover of size K.*

(2) *The reduction is planarity-preserving and weakly parsimonious.*

Proof. Part 1. Observe that for any vertex cover, one needs to pick at least half of the nodes from each cycle, two of the three nodes from each triangle, and one from the

simple edge for each two-literal clause. (Recall that there are $3m$ two-literal clauses in f'.) The sum is exactly K. Given this observation, the proof is similar to the one given in [5].

Part 2. It can be easily verified by observing in Figure 4.2 that the reduction is planarity-preserving. To see that for each distinct satisfying assignment of f', there are 2^m distinct vertex covers of size K, note that, by Lemma 4.1, for each satisfying assignment, all but one clause in each clause group has only one true literal. This forces the choice of vertices from the clause graph for all clauses having only one true literal. For each satisfying assignment and for each clause group there is one clause in the clause group in which both the literals are true. For each such clause any of the two vertices can be included in the vertex cover. Since there are m clause groups, we have m such clauses and hence we have 2^m different vertex covers for each satisfying assignment. □

Note that our reduction shows that counting the number of vertex covers of size $\leq k$ is #P-hard even if there are *no* vertex covers of size strictly less than k. For the next two results, we use such an instance of #-vertex cover for our reductions.

(2) Minimum Dominating Set: The reduction is from the Minimum Vertex Cover problem. The reduction in [13] from Vertex Cover to Dominating Set can easily be modified to get a parsimonious reduction. Let $G_1 = (V_1, E_1)$, $V_1 = \{v_1, \ldots, v_n\}$ be an instance of the Minimum Vertex Cover problem. We construct an instance $G_2 = (V_2, E_2)$ of the Minimum Dominating Set problem as follows. There is one vertex in V_2 corresponding to every vertex in V_1. For each edge in G_1 we also introduce two additional vertices and join them to the two endpoints of the original edge. Formally, $V_2 = U_1 \cup U_2$ and $E_2 = A_1 \cup A_2$, where

$$U_1 = \{u_i \mid v_i \in V_1\},$$

$$U_2 = \{x_1^{ij}, x_2^{ij} \mid (v_i, v_j) \in E_1\},$$

$$A_1 = \{(u_i, u_j) \mid (v_i, v_j) \in E_1\},$$

$$A_2 = \{(u_i, x_1^{ij}), (x_1^{ij}, u_j), (u_i, x_2^{ij}), (x_2^{ij}, u_j) \mid (v_i, v_j) \in E_1\}.$$

Claim 4.4. *G_1 has a minimum vertex cover of size k if and only if G_2 has a minimum dominating set of size k. Furthermore, the reduction is planarity-preserving and parsimonious.*

Proof. It is easy to see that the reduction is planarity-preserving. Consider a minimum vertex cover $VC = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ of G_1. Corresponding to VC we claim that there is exactly one dominating set in the graph G_2, namely, the vertex set $DS = \{u_{i_1}, u_{i_2}, \ldots, u_{i_k}\}$. First note that for each edge in the original graph G we have four new edges and two new vertices in G_2. Consider a pair of nodes of the form x_1^{ij}, x_2^{ij} connected to the nodes u_i and u_j. It is clear that the only way to dominate both x_1^{ij}, x_2^{ij} by using only one node is to include one of u_i or u_j in the dominating set. We need to consider two cases. First consider the case when exactly one of v_i or v_j is in VC. Then it is clear that there is exactly one dominating set DS in G_2 corresponding to VC. When both v_i and v_j are in VC, the minimality of VC implies that at least one edge incident on v_i and at least one edge incident on v_j are covered solely by v_i and v_j, respectively. This implies that both u_i and u_j have to be in

any feasible dominating set of G_2. Thus we have exactly one dominating set DS in G_2 corresponding to the vertex cover VC in G_1. Conversely, consider a minimum dominating set $DS = \{u_{i_1}, u_{i_2}, \ldots, u_{i_k}\}$ of size k in G_2. Consider an edge (v_i, v_j) in G. If u_i and u_j are not in DS, then by construction of G_2, both x_1^{ij} and x_2^{ij} are in DS. But we could then construct a new dominating set DS' of G_2, where

$$DS' = DS - \{x_1^{ij}, x_2^{ij}\} \cup \{u_i\}.$$

Clearly $|DS'| < |DS|$, which is a contradiction to the assumption that DS is a minimum dominating set. Thus, DS does not contain any vertex from the set U_2. We now claim that $VC = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ is a vertex cover of G. The claim follows by observing that corresponding to each edge $(v_i, v_j) \in E_1$, at least one of the vertices $u_i, u_j, x_1^{ij}, x_2^{ij}$ are in the set DS. We have already argued that $x_1^{ij}, x_2^{ij} \notin DS$. Thus one of u_i, u_j is in DS. The corresponding vertex in VC is seen to cover the edge (v_i, v_j). □

(3) Feedback Vertex Set: The reduction is from the Minimum Vertex Cover problem. Starting from an instance $G_1(V_1, E_1)$ of the Minimum Vertex Cover problem, we construct the graph G_2 that is identical to the one given for the Minimum Dominating Set problem. By arguments similar to those given in the proof of the Minimum Dominating Set problem, it is easy to see that the G_1 has a minimum vertex set of size K if and only if G_2 has a feedback vertex set of size K and the reduction is parsimonious.

(4) Subgraph Isomorphism: This follows directly by taking the graph H to be a simple cycle on n nodes, and the weakly parsimonious reduction from 3Sat to the Hamiltonian Circuit problem given in Provan [20]. □

Corollary 4.5. *The problems* #Pl-Minimum Vertex Cover, #Pl-Minimum Dominating Set, #Pl-Minimum Feedback Vertex Set, *and* #Pl-Subgraph Isomorphism *are* #P*-complete.*

4.4. Parsimonious reductions and other graph problems. In this section we briefly discuss why the reductions studied by [6, 15] and [3, 4] from X3C to various other graph problems are parsimonious and planarity-preserving.

Theorem 4.6. *There exist planarity-preserving and parsimonious reductions from* X3C *to each of the problems* (1) Minimum Clique Cover, (2) Partition into Claws, (3) Bipartite Dominating Set, (4) Partition into Triangles, *and* (5) Minimum Hitting Set.

Proof. (1) Minimum Clique Cover: The reduction is the same as given in [3, 5]. Given an instance $I(X, C)$ of X3C such that $|X| = 3p$ and $|C| = m$, we construct an instance G of the Minimum Clique Cover problem such that G has a clique cover with cliques of size 3 if and only if I has a solution. The reduction consists of replacing each triple in the instance of I by a triangle and by replacing an edge from a triple to an element by a set of triangles. The reduction is illustrated in Figure 4.3. Formally, for each element, we have a vertex in G. Corresponding to each triple $t_i = \{x_i, y_i, z_i\}$ and the associated edges $(t_i, x_i), (t_i, y_i), (t_i, z_i)$, we create the subgraph as shown in Figure 4.3. The graph G obtained by carrying out the above reduction for each triple has $3p+9m$ vertices and $18m$ edges. The reduction is planarity-preserving, as each component is planar and they are joined in a planarity-preserving way. We claim that I has a solution if and only if G has a clique cover of size $(p + 3m)$. In particular, as shown in [3, 5], if $t_1, \ldots, t_p$ is the set of triples in an exact cover, then

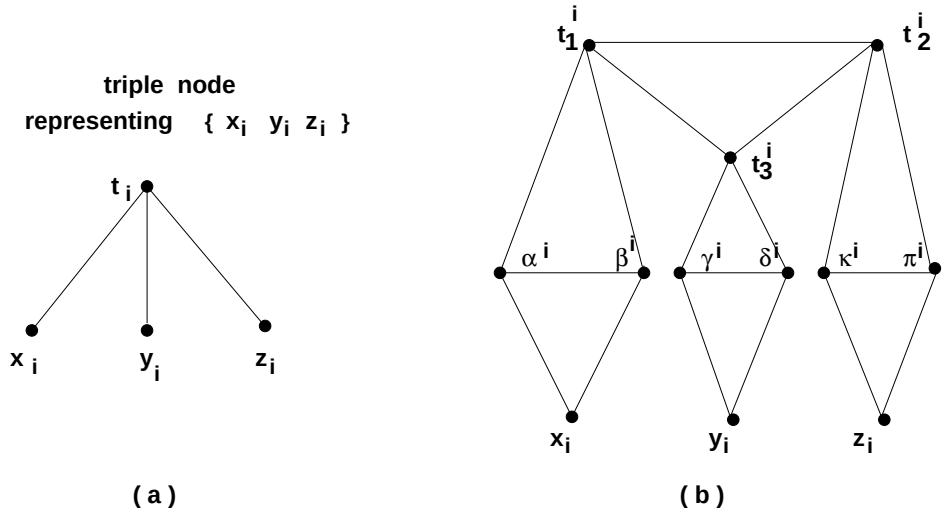


FIG. 4.3. *Figure illustrating the reduction from* X3C *to* CLIQUE COVER.

the corresponding clique cover is constructed by taking

$$\{\alpha^i, \beta^i, x_i\}, \{\gamma^i, \delta^i, y_i\}, \{\kappa^i, \pi^i, z_i\}, \{t_1^i, t_2^i, t_3^i\}$$

whenever $t_i = \{x_i, y_i, z_i\}$ is in the exact cover and by taking the cliques

$$\{\alpha^i, \beta^i, t_1^i\}, \{\gamma^i, \delta^i, t_2^i\}, \{\kappa^i, \pi^i, t_3^i\}$$

when the corresponding triple t_i is not in the exact cover. Conversely, since G has $3p + 9m$ vertices, if G has a clique cover of size $p + 3m$ it implies that each clique consists of exactly three vertices. (Recall that we do not have cliques of size 4 in G.) The corresponding exact cover is given by choosing those $t_i \in C$ such that the triangles t_1^i, t_2^i, t_3^i are in the clique cover. Finally, we prove that the reduction is parsimonious. First note that if the triple triangle $\{t_1^i, t_2^i, t_3^i\}$ is not chosen then we have to choose the triangles $\{\alpha^i, \beta^i, t_1^i\}, \{\gamma^i, \delta^i, t_2^i\}, \{\kappa^i, \pi^i, t_3^i\}$ so as to cover the triple vertices. Second, once the triangle $\{t_1^i, t_2^i, t_3^i\}$ is chosen there is exactly one way for the auxiliary nodes (and thus the element nodes) to be covered, namely, the lower triangle corresponding to the covering triple, i.e., choosing the triangles

$$\{\alpha^i, \beta^i, x_i\}, \{\gamma^i, \delta^i, y_i\}, \{\kappa^i, \pi^i, z_i\}.$$

These observations immediately imply that the reduction is parsimonious.

(2) PARTITION INTO CLAWS: The reduction is from X3C and is the same as the one given in [3]. The reduction consists of the following steps.

1. Construct the bipartite graph $G(C \cup X, E)$ corresponding to the given instance $I(X, C)$ of X3C.

2. As in [3], we assume that each element vertex appears in either two or three sets; i.e., the element vertices have a degree 2 or 3.

3. For each element of degree 3, we add an extra edge and for each element of degree 2, we add two extra edges. This is shown in Figure 4.4. Let G_1 denote the resulting graph.

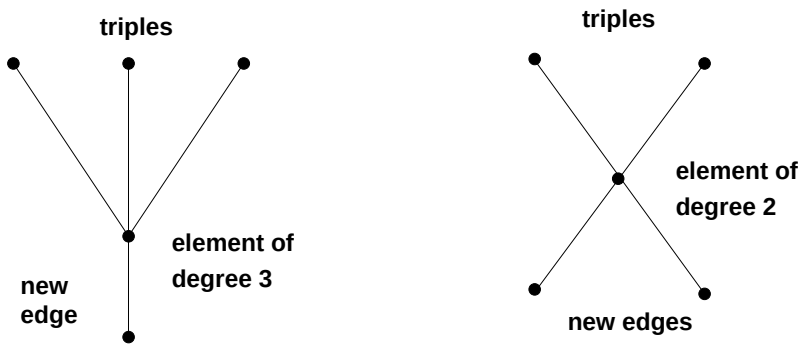


FIG. 4.4. *Figure illustrating the reduction from* X3C *to* PARTITION INTO CLAWS.

Clearly the reduction is planarity-preserving. We now recall the proof in [3] to show that the edges of G_1 can be partitioned into a disjoint set of claws. Note that each element vertex is adjacent to either 1 or 2 vertices of degree 1, so it follows that each element node must be the center of at least one claw. But each such element node has degree 4 and hence can be the center of exactly one claw. After removing the claws from G_1 the resulting graph G_2 has the property that all element nodes have degree 1. This implies that the only way to partition G_2 into claws is for each triple to have a degree of 0 or 3. Thus the triples with degree 3 induce a solution for the X3C in an obvious way. Conversely, given a solution for I, the above argument can be reversed to yield a partition of the edges in G_1 into claws. The following observations immediately imply that the reduction is parsimonious.

1. There is a unique way to pick the claws in G_1 with the element nodes as centers.

2. In G_2, each triple vertex has degree 3 or 0 and each element node has degree 1 in G_2.

(3) BIPARTITE DOMINATING SET: Reduction from X3C. The construction is similar to that in [3]. Let $I(X, C)$ be an instance of PL-X3C with each element occurring in at most three triples. We first construct the bipartite graph G associated with I. Next, we attach a 2-claw ($K_{1,2}$) to each triple vertex in G as shown in Figure 4.5. (In [3], they add a path of length 2.) Let G' denote the graph obtained as a result of the transformation. The construction is depicted in Figure 4.5. Since G is bipartite and we added a claw as shown in Figure 4.5, it follows that G' is also bipartite. Also note that the reduction is planarity-preserving and thus G' is planar. Let the number of triples be m and the number of elements be $3p$. Then we set $k = p + m$. Now by arguments similar to those in [3], it is easy to see that G' has a dominating set of size k if and only if I has an exact cover of size p. We prove the reduction is parsimonious. Consider a solution $S(I)$ for I. Since each triple in the solution covers three distinct element nodes, these element nodes cannot be used to dominate the vertices in G' without increasing the cardinality of the solution for G'. This means that for each of the p triples chosen in the solution $S(I)$, we have exactly one node in G' that can be used in the dominating set so as to dominate all the element nodes. Moreover, due to the constraints on the size of the dominating set in G', it follows that we can select exactly one vertex per claw (the vertex with degree 3 and marked a) in the dominating set. These observations imply that the reduction is parsimonious.

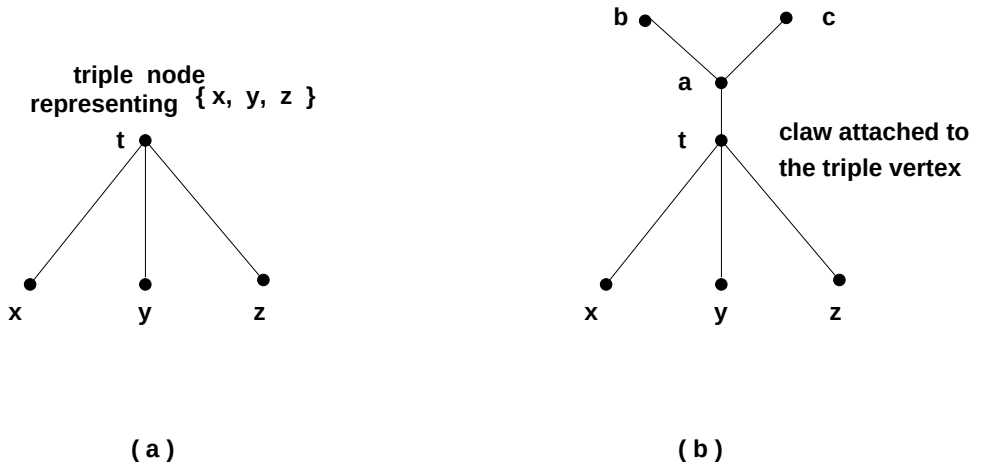


FIG. 4.5. *Figure illustrating the parsimonious reduction from* X3C *to* BIPARTITE DOMINATING SET. *It is easy to see that the reduction preserves planarity of the graph.*

(4) PARTITION INTO TRIANGLES: The reduction is from X3C and is the same as the reduction described in the proof of CLIQUE COVER. Given that the resulting graph has no cliques of size 4, the proof follows.

(5) MINIMUM HITTING SET: As given in [5], each instance of vertex cover can be seen to be an instance of a hitting set, in which every edge (u, v) corresponds to the set $\{u, v\}$. The elements of the set are simply the nodes of the graph. The result now follows by noting that there is a weakly parsimonious reduction from 3SAT to MINIMUM VERTEX COVER. □

COROLLARY 4.7. *The problems* #PL-MINIMUM CLIQUE COVER, #PL-PARTITION INTO CLAWS, #PL-BIPARTITE DOMINATING SET, #PL-PARTITION INTO TRIANGLES, *and* #PL-MINIMUM HITTING SET *are* #P*-complete.*

THEOREM 4.8. *Let* Π *be one of the problems in Table* 1.1. *It is* #P*-complete to count the number of solutions to* Π*, even when one is given an instance of* Π *and a solution which is guaranteed to satisfy* Π.

Proof. Starting from a 3CNF formula obtained in the proof of Theorem 3.4, we now do the same set of reductions discussed in the earlier theorems to obtain an instance of the problem Π. Since we know that the 3CNF formula is satisfiable, it follows that the instance of Π has a solution. □

5. Unique and ambiguous planar problems. Our parsimonious planar crossover box for 3SAT can also be used to show that additional problems for planar CNF formulas are *as hard as* the corresponding problems for arbitrary CNF formulas, with respect to polynomial time or random polynomial time reducibilities. We briefly describe these results. We first recall the definitions of D^P and random polynomial time reductions from [19, 27].

DEFINITION 5.1. $\mathsf{D}^P = \{L_1 - L_2 \mid L_1, L_2 \in \mathsf{NP}\}$. *Intuitively, a problem is in* D^P *if it can be solved by asking one question in* NP *and one question in* Co-NP.

DEFINITION 5.2. *Problem A is reducible to problem B by a randomized polynomial time reduction if there is a randomized polynomial time Turing machine T and a polynomial p such that*

(1) $\forall x,\ [x \notin A \rightarrow T[x] \notin B]$;

(2) $\forall x,\ [x \in A \rightarrow T[x] \in B$ *with probability at least* $1/p(|x|)]$. □

THEOREM 5.3. UNIQUE-PL-3SAT *is* D^P*-complete under randomized polynomial time reductions.*

Proof. We modify the proof of the D^P-completeness of UNIQUE SAT in [27], so that whenever their reduction outputs a boolean formula f, our reduction outputs the planar formula $\text{PL}(f)$ obtained by applying the parsimonious planar crossover box to f. The formula $\text{PL}(f)$ has exactly the same number of satisfying assignments as f. In particular, $\text{PL}(f)$ is uniquely satisfiable if and only if f is uniquely satisfiable. □

A second example is the following.

THEOREM 5.4. AMBIGUOUS-PL-3SAT *is* NP*-complete.*

Proof. Given an instance of an arbitrary 3CNF formula f, we first construct a new formula using the same construction as in Step 1 of Theorem 3.4. As pointed out in the proof of Theorem 3.4, the new formula is ambiguously satisfiable if and only if the original formula is satisfiable. We then do the same sequence of reductions as in Theorem 3.4 to obtain a planar formula that is ambiguously satisfied if and only if the original formula is satisfied. □

Using the ideas similar to those in the proof of Theorem 3.5, we can prove the following theorem.

THEOREM 5.5. AMBIGUOUS-1-VALID PL-3SAT *is* NP*-complete.*

COROLLARY 5.6. UNIQUE-1-VALID PL-3SAT *is* Co-NP*-complete.*

Proof. To prove the membership in Co-NP, consider an arbitrary formula $F(x_1, \ldots, x_n)$, which is an instance of 1-VALID PL-3SAT. By the definition of 1-VALID formulas, an assignment $\mathbf{v}$ to the variables such that $\mathbf{v}[x_1] = \mathbf{v}[x_2] = \cdots \mathbf{v}[x_n] = 1$ satisfies $F(x_1, \ldots, x_n)$. Now consider the formula $H(x_1, \ldots, x_n) = F(x_1, x_2, \ldots, x_n) \wedge (\overline{x_1} \vee \overline{x_2} \ldots \overline{x_n})$. F is uniquely satisfiable if and only if H is unsatisfiable. To prove Co-NP-hardness, given a formula $f(x_1, x_2, \ldots, x_n)$, we construct a formula g such that

$$g(x_1, \ldots, x_{n+1}) = [f(x_1, x_2, \ldots, x_n) \wedge x_{n+1}] \bigvee (\overline{x_1} \wedge \overline{x_2} \ldots \overline{x_{n+1}}).$$

Now using ideas similar to those in the proof of Theorem 3.5, we obtain a planar formula g_1 with the following properties:

(1) g_1 is 1-valid.

(2) g_1 is uniquely satisfiable *if and only if* f is unsatisfiable. □

Combining our parsimonious planar crossover box for 3SAT and the reductions to prove Theorem 3.8, we get that exact analogues of Theorems 5.3–5.5 hold for each of the problems EX3SAT, 1-3SAT, 1-EX3SAT, and 1-EX3MONOSAT. Thus, we have the following corollary.

COROLLARY 5.7. *Let* Π *be one of the following problems:* EX3SAT, 1-3SAT, 1-EX3SAT*, or* 1-EX3MONOSAT. *Then the problem* AMBIGUOUS-PL-Π *is* NP*-complete and the problem* UNIQUE-PL-Π *is* D^P*-complete under randomized polynomial time reductions.*

As a corollary of our parsimonious reductions, the unique versions of many graph problems are also D^P-complete.

COROLLARY 5.8. *Let* Π *be one of the problems* PL-PARTITION INTO TRIANGLES, PARTITION INTO CLAWS*, or* BIPARTITE DOMINATING SET. *Then the problem* AMBIGUOUS-Π *is* NP*-complete and the problem* UNIQUE-Π *is* D^P*-complete under randomized polynomial reductions.*

Proof. Given that each reduction in the sequence of reductions 3SAT $\rightarrow$ PL-3SAT $\rightarrow$ PL-EX1-3SAT $\rightarrow$ PL-X3C is parsimonious, that UNIQUE 3SAT is D^P-complete,

and that the reduction from X3C to each of the problems mentioned above is parsimonious, the proof of the corollary is similar to the proof of Theorem 5.4. □

5.1. Nonapproximability results for integer linear programming. Next, we give an application of our result that AMBIGUOUS PL-3SAT is NP-complete and prove that it is not possible to approximate the optimal value of the objective function of an integer linear program.

An instance of an integer linear program consists of a system of linear inequalities and an objective function which is to be maximized (minimized); i.e., maximize (minimize) $\mathbf{c}x$ subject to the constraints $\mathbf{A}x \leq \mathbf{b}$. The variables x are allowed to take only integer values. We say that a minimization problem Π is ϵ-approximable, $\epsilon > 1$ (or has an ϵ-approximation) if there is a polynomial time algorithm that, given an instance $I \in \Pi$, finds a solution which is within a factor ϵ of an optimal solution for I.

THEOREM 5.9. *Unless* P = NP, *given an instance of the integer linear program problem and a feasible solution, the maximum (minimum) value of the objective function is not polynomial time ϵ-approximable for any $\epsilon > 1$, even when the bipartite graph associated with the set of constraints is planar.*

Proof. We prove the theorem for the maximization version of the problem. The proof for the minimization version is similar and hence omitted.

Step 1. Given a 3SAT formula $f(x_1, x_2, \ldots, x_n)$, we construct a formula

$$g(x_1, \ldots, x_{n+1}) = [f(x_1, x_2, \ldots, x_n) \wedge x_{n+1}] \vee (\overline{x_1} \wedge \overline{x_2} \ldots \overline{x_{n+1}}).$$

It follows that for any assignment $\mathbf{v}$, $\mathbf{v}[g(x_1, \ldots, x_{n+1})] = 1$ if and only if either (i) $\mathbf{v}[f(x_1, x_2, \ldots, x_n)] = 1$ and $\mathbf{v}[x_{n+1}] = 1$, *or* (ii) $\mathbf{v}[x_1] = \mathbf{v}[x_2] = \cdots \mathbf{v}[x_{n+1}] = 0$.

Step 2. Starting from $g(x_1, \ldots, x_{n+1})$, we construct a PL-3SAT formula $\hat{g}(x_1, x_2, \ldots, x_{n+1}, t_1, \ldots, t_m)$ such that $\hat{g}(x_1, x_2, \ldots, x_{n+1}, t_1, \ldots, t_m)$ is satisfiable if and only if $g(x_1, x_2, \ldots, x_{n+1})$ is satisfiable.

The construction can be carried out as in Step 2 in the proof of Theorem 3.4. We therefore omit the details here.

Step 3. Let $\hat{g} = G_1 \wedge G_2 \ldots G_r$. Construct a new 1-EX3-MONOSAT formula h from $\hat{g}$ such that $\hat{g}$ is satisfiable if and only if h is satisfiable. Let $h = C_1 \wedge C_2 \ldots C_p$. Replace each clause $C_i = (x_{i_1} + x_{i_2} + x_{i_3})$ by the inequality $(x_{i_1} + x_{i_2} + x_{i_3}) \geq 1$. All the inequalities corresponding to the clauses make up the constraints. We also add constraints that $\forall i,\ x_i \in \{0, 1\}$. The objective function is now simply x_{n+1}. It is easy to verify that the maximum value of the objective function is exactly 1 if $f(x_1, x_2, \ldots, x_n)$ is satisfiable and is 0 otherwise. Hence it follows that unless P = NP the integer linear program problem has no polynomial time ϵ-approximation algorithm for any $\epsilon > 1$. □

6. Conclusions and open problems. We showed that for many problems Π studied in the literature, the problems #PL-Π, AMBIGUOUS-PL-Π, and UNIQUE-PL-Π are *as hard as* the respective problems #Π, AMBIGUOUS-Π, and UNIQUE-Π with respect to polynomial time or random polynomial time reducibilities. We note that the problem #PL-HAMILTONIAN-CYCLE was proved to be #P-complete by Provan [20]. We can give an alternate proof of the #P-hardness of #PL-HAMILTONIAN-CYCLE by a reduction from a variant of $RED1$. The reduction is significantly more complicated than that in [20]. Consequently, we omit it here.

As corollaries of our results, we have shown that many planar problems are complete for the classes NP, #P, and D^P. Our results and their proofs provide the following general tools for proving hardness results for planar problems.

1. We have shown how parsimonious and weakly parsimonious crossover boxes can be used to prove the #P-hardness of many planar counting problems. These ideas were used to prove the #P-hardness of problem #1-Valid Pl-3Sat.

2. We extended the class of basic planar CNF satisfiability problems that are known to be NP-complete. Previously, only Pl-3Sat [15] and Pl-1-Ex3Sat [3] were known to be NP-hard. We expect that the variants of the problem Pl-3Sat shown to be NP-hard here will be useful in proving hardness results for many additional planar problems. In particular, we have already shown that the problem Pl-1-3MonoSat and its variant *RED*1 are especially useful in proving the #P-hardness of many planar graph problems.

3. We have shown that the problem Ambiguous-Pl-3Sat can be used to prove the nonapproximability of linear integer programming. Recently, there has been a lot of research in the area of approximability of graph and combinatorial problems, and the tools for showing negative results are few. Our proof of the nonapproximability of the minimum or maximum objective value of an integer linear program is direct and significantly different from the proof given in Kann [12] or Zuckerman [28]. Moreover, in [9], we show how to use the NP-completeness of Ambiguous-Pl-3Sat to show the nonapproximability of several constrained optimization problems even when restricted to *planar instances*. (The results in [12, 28] do not hold for planar instances.)

Finally, the results presented here and their proofs suggest a number of open problems including the following.

1. Can natural planar problems be found that are complete for additional complexity classes such as PSPACE, #PSPACE, MAX SNP, MAX Π_1, etc.? (In recent papers [8, 9, 17, 10], we partially answer this question by showing that a number of problems are complete for the classes PSPACE, #PSPACE, MAX-SNP, MAX Π_1, even when restricted to planar instances.)

2. Valiant [25] has shown that the problem #2Sat is #P-complete. How hard is the problem #Pl-2Sat? We conjecture that the problem is #P-complete, but it seems to us that different techniques than the ones used here are required to prove this. Recently Vadhan has affirmatively proved the conjecture [29].

3. We have shown that many unique satisfiability problems are complete for D^P, even when restricted to planar instances. Using our parsimonious reductions, we then proved the D^P-completeness of a number of graph problems for planar graphs. A number of such problems for planar graphs remain open. For example, how hard is the problem Unique-Pl-Hamiltonian Circuit?

4. Do results similar to the ones proved in this paper hold for other restricted classes of graphs, e.g., intersection graphs of unit disks and squares? Such graphs have been studied extensively by [2, 7, 18] in the context of image processing, VLSI design, geometric location theory, and network design.

Acknowledgments. We thank the anonymous referee for invaluable suggestions. These suggestions significantly improved the quality of presentation and helped us in correcting a number of errors in the earlier draft. We thank Salil Vadhan for providing a draft of his paper [29].

REFERENCES

[1] G. Brightwell and P. Winkler, *Counting linear extensions is #P-complete*, in Proc. 23rd Annual ACM Symposium on Theory of Computing, 1991, pp. 175–181.

[2] B.N. Clark, C.J. Colbourn, and D.S. Johnson, *Unit disk graphs*, Discrete Math., 86 (1990), pp. 165–177.

[3] M.E. Dyer and A.M. Frieze, *Planar 3DM is NP-complete*, J. Algorithms, 7 (1986), pp. 174–184.

[4] M.E. Dyer and A.M. Frieze, *The complexity of partitioning graphs into connected components*, Discrete Appl. Math., 10 (1985), pp. 139–153.
[5] M.R. Garey and D.S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.
[6] M.R. Garey, D.S. Johnson, and L. Stockmeyer, *Simplified NP-complete problems*, Theoret. Comput. Sci., 1 (1976), pp. 237–267.
[7] D.S. Hochbaum and W. Maass, *Approximation schemes for covering and packing problems in image processing and VLSI*, J. Assoc. Comput. Mach., 32 (1985), pp 130–136.
[8] H. B. Hunt III, R. E. Stearns, and M. V. Marathe, *Local Reductions, Generalized Satisfiability, Complexity and Efficient Approximability*: I, in preparation. A preliminary version of the paper appears in Proc. 9th ACM Conference on Structure in Complexity Theory, 1994, pp. 356–366.
[9] H. B. Hunt III, R. E. Stearns, and M. V. Marathe, *Local Reductions, Generalized Satisfiability, Complexity and Efficient Approximability*: II, in preparation. A preliminary version of the paper appears in Proc. 9th ACM Conf. on Structure in Complexity Theory, 1994, pp. 356–366.
[10] H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, S. S. Ravi, D. J. Rosenkrantz, and R.E. Stearns, *Parallel approximation schemes for planar and near-planar satisfiability and graph problems*, Inform. and Comput., submitted, 1998. A preliminary version appears in Proc. 14th Foundations of Software Technology and Theoretical Computer Science (FST & TCS), Lecture Notes in Comp. Sci. 761, Springer-Verlag, New York, 1994, pp. 342–353. A complete version of the paper appears as Technical Report No. LA-UR-96-2723, Los Alamos National Laboratory, Los Alamos, NM, 1996.
[11] H.B. Hunt III, V. Radhakrishnan, M.V. Marathe, and R.E. Stearns, *On the Complexity of Planar Graph/Hypergraph Coloring*, in preparation, 1997.
[12] V. Kann, *Polynomially bounded minimization problems which are hard to approximate*, Nordic J. Comput., 1 (1994), pp. 317–331
[13] R.M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.
[14] R. Karp and M. Luby, *Monte Carlo algorithms for enumeration and reliability problems*, in Proc. 24th ACM Symp. on Theory of Computing, 1983, pp 54–64. A complete version appears in J. Complexity, 1 (1985), pp. 45–64.
[15] D. Lichtenstein, *Planar formulae and their uses*, SIAM J. Comput., 11 (1982), pp. 329–343.
[16] N. Linial, *Hard enumeration problems in geometry and combinatorics*, SIAM J. Algebraic Discrete Meth., 7 (1986), pp. 331–335.
[17] M.V. Marathe, H.B. Hunt III, R.E. Stearns, and V. Radhakrishnan, *Complexity of Hierarchically and* 1*-Dimensional Periodically Specified Problems*, Technical Report No. LA-UR-95-3348, Los Alamos National Laboratory, Los Alamos, NM, August 1995. Also presented at the *DIMACS Workshop on Satisfiability Problem: Theory and Applications,* March 1996.
[18] N. Meggido and K Supowit, *On the complexity of some common geometric location problems*, SIAM J. Comput., 13 (1984), pp. 182–196.
[19] C. Papadimitriou, *Computational Complexity,* Addison-Wesley, Reading, MA, 1994.
[20] J.S. Provan, *The complexity of reliability computations in planar and acyclic graphs*, SIAM J. Comput., 15 (1986), pp. 694–702.
[21] J.S. Provan and M. O. Ball, *On the complexity of counting cuts and of computing the probability that a graph is connected*, SIAM J. Comput., 12 (1983), pp. 777–788.
[22] S. Saluja, K. V. Subramanium, and M. Thakur, *Descriptive complexity of #P functions*, J. Comput. System Sci., 50 (1995), pp. 493–505.
[23] J.B. Saxe, *Two Papers on Graph Embedding Problems*, Technical Report CMU-CS-80-102, Dept. of Comp. Science, Carnegie Mellon University, 1980.
[24] T.J. Schaefer, *The complexity of satisfiability problems*, in Proc. 10th ACM Symp. on Theory of Computing, 1978, pp. 216–226.
[25] L.G. Valiant, *The complexity of enumeration and reliability problems*, SIAM J. Comput., 8 (1979), pp. 410–421.
[26] L.G. Valiant, *The complexity of computing the permanent*, Theoret. Comput. Sci., 8 (1979), pp. 189–201.
[27] L. G. Valiant and V. V. Vazirani, *NP is as easy as detecting unique solutions*, in Proc. 17th Annual ACM Symp. on Theory of Computing, 1985, pp. 458–463.
[28] D. Zuckerman, *NP-complete problems have versions that are hard to approximate*, in Proc. 8th Annual ACM Conf. on Structure in Complexity Theory, 1993, pp. 305–312. A complete version titled *On unapproximable versions of NP-complete problems* appears in SIAM J. Comput., 25 (1996), pp. 1293–1304.
[29] S. Vadhan, *The Complexity of Counting in Sparse, Regular and Planar Graphs*, Technical Report, MIT, Cambridge, MA, 1997. Also available online from www-math.mit.edu/∼salil.

SIAM J. COMPUT.
Vol. 27, No. 4, pp. 1168–1189, August 1998
© 1998 Society for Industrial and Applied Mathematics
013

GUARANTEEING FAIR SERVICE TO PERSISTENT DEPENDENT TASKS*

AMOTZ BAR-NOY†, ALAIN MAYER‡, BARUCH SCHIEBER§, AND MADHU SUDAN¶

Abstract. We introduce a new scheduling problem that is motivated by applications in the area of access and flow control in high-speed and wireless networks. An instance of the problem consists of a set of *persistent tasks* that have to be scheduled repeatedly. Each task has a demand to be scheduled "as often as possible." There is no *explicit* limit on the number of tasks that can be scheduled concurrently. However, such limits are imposed implicitly because some tasks may be in conflict and cannot be scheduled simultaneously. These conflicts are presented in the form of a conflict graph. We define parameters which quantify the *fairness* and *regularity* of a given schedule. We then proceed to show lower bounds on these parameters and present fair and efficient scheduling algorithms for the case where the conflict graph is an interval graph. Some of the results presented here extend to the case of perfect graphs and circular-arc graphs as well.

Key words. fairness, dining philosophers problem, scheduling, interval graphs

AMS subject classifications. 90B35, 68M20, 68Q20, 68Q25

PII. S0097539795282092

1. Introduction.

In this paper we consider a new form of a scheduling problem that is characterized by the following two features.

Persistence. A task does not simply terminate once it is scheduled. Instead, each task must be scheduled infinitely many times. The goal is to schedule every task as frequently as possible.

Dependency. Some tasks conflict with each other and hence cannot be scheduled concurrently. These conflicts are represented by a conflict graph. This graph imposes constraints on the sets of tasks that may be scheduled concurrently. Note that these constraints are not based simply on the cardinality of the sets but rather on the identity of the tasks within the sets.

We consider both the problems of *allocation*, i.e., how often should a task be scheduled and *regularity*, i.e., how evenly spaced are lengths of the intervals between successive schedulings of a specific task. We present a more formal description of this problem next and discuss our primary motivation immediately afterward. While all our definitions are presented for general conflict graphs, our applications, bounds, and algorithms are for special subclasses—perfect graphs, interval graphs, and circular arc graphs.

*Received by the editors February 24, 1995; accepted for publication (in revised form) June 13, 1996; published electronically May 19, 1998.

http://www.siam.org/journals/sicomp/27-4/28209.html

†Dept. of Electrical Engineering, Tel Aviv University, Tel Aviv, Israel 69978 (amotz@eng.tau.ac.il). Part of this work was done while the author was at the IBM T. J. Watson Research Center.

‡Bell Laboratories, Lucent Technologies, 700 Mountain Ave., Murray Hill, NJ 07974 (alain@research.bell-labs.com). Part of this work was done while the author was at the IBM T. J. Watson Research Center, and this work was partially supported by an IBM Graduate Fellowship, NSF grant CCR-93-16209, and CISE Institutional Infrastructure grant CDA-90-24735.

§IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY 10598 (sbar@watson.ibm.com).

¶Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139-3594 (madhu@lcs.mit.edu).

Problem statement. An instance of the scheduling problem consists of a conflict graph G with n vertices. The vertices of G are the tasks to be scheduled, and the edges of G define pairs of tasks that cannot be scheduled concurrently. The output of the scheduling algorithm is an infinite sequence of subsets of the vertices, $I_1, I_2, \ldots$, where I_t lists the tasks that are scheduled at time t. Note that I_t must be an independent set of G for all t.

In the form above, it is hard to analyze the running time of the scheduling algorithm. We consider instead a finite version of the above problem and use it to analyze the running time.

Input. *A conflict graph G and a time t.*

Output. *An independent set I_t denoting the set of tasks scheduled at time unit t.*

The objective of the scheduling algorithm is to achieve a *fair* allocation and a *regular* schedule. We next give some motivation and describe the context of our work. As we will see, none of the existing measures can appropriately capture the "goodness" of a schedule in our framework. Hence we proceed to introduce measures which allow a better presentation of our results.

1.1. Motivation.

Session scheduling in high-speed local-area networks. The main motivation for this work arises from the session scheduling problem on a network called MetaRing. MetaRing [CO93] is a recent high-speed local-area ring network that allows "spatial bandwidth reuse"; i.e., in contrast to other ring networks which may only allow one source destination pair to communicate on the ring at a time, the MetaRing can allow several such pairs to communicate with each other, provided the pairs do not use the same link for the communication. This concurrent access and transmission of user sessions is implemented using only minimal intermediate buffering of packets. A typical use of the MetaRing network involves several users trying to establish "sessions." A session is simply a source destination pair, where the source wishes to send data over to the destination. The data are generated over time and need to be shipped over at regular intervals, if not immediately. In the general setting, the set of sessions can change dynamically. We restrict our attention to the static case. This restriction is justified by the fact that sessions typically last for a long while. As a result of the minimal intermediate buffering, a session can send its data over only if it has exclusive use of all the links in its route. Consequently, sessions whose routes share a link are in conflict. These conflicts must be regulated by breaking the data sent in a session into units of quotas that are transmitted according to some schedule. This schedule has to be *efficient* and *fair*. Efficient means that the total number of quotas transmitted (throughput) is maximized. Fair means that the throughput of *each* session is maximized, and that the time between successive activation of a session is minimized, so that large buffers at the source nodes can be avoided. It has been recognized [CCO93] that the access and flow control in such a network should depend on locality in the conflict graph. However, no firm theoretical basis for an algorithmic framework has been proposed up to now. To express this problem as our scheduling problem we create a circular-arc graph, the vertices of which are the sessions, and in which vertices are adjacent if the corresponding paths associated with the sessions intersect. Since an independent set in this graph is a collection of mutually nonconflicting sessions, they can all communicate simultaneously. Thus a schedule is a sequence of independent sets in this graph. Our goal will be to produce such a schedule which is efficient and fair, in a sense to be made precise later.

Time sharing in wireless networks. A second naturally occurring scenario where our model can be applied is in the scheduling of base-station transmissions in a cellular network. We now describe this setting in detail. Most indoor designs of wireless networks are based on a *cellular* architecture with a very small cell size (see, e.g., [Goo90].) The cellular architecture comprises two levels—a stationary level and a mobile level. The stationary level consists of fixed base stations that are interconnected through a backbone network. The mobile level consists of mobile units that communicate with the base stations via wireless links. The geographic area within which mobile units can communicate with a particular base station is referred to as a cell. Neighboring cells overlap with each other, thus ensuring continuity of communications. The mobile units communicate among themselves, as well as with the fixed information networks, through the base stations and the backbone network. The continuity of communications is a crucial issue in such networks. A mobile user who crosses boundaries of cells should be able to continue its communication via the new base station. To ensure this, base stations periodically need to transmit their identity using the wireless communication. In some implementations the wireless links use infrared waves. Therefore, two base stations, the cells of which overlap, are in conflict and cannot transmit their identity simultaneously. These conflicts have to be regulated by a time-sharing scheme. This time sharing has to be *efficient* and *fair*. Efficient means that the scheme should accommodate the maximal number of base stations. Fair means that the time between two consecutive transmissions of the same base station should be less than the time it takes a user to cross its corresponding cell. Once again this problem can be posed as our graph scheduling problem in which the vertices of the graph are the base stations and an edge indicates that the base stations belong to overlapping cells. Independent sets again represent lack of conflict and a schedule will thus be a sequence of independent sets.

1.2. Relationship to past work. Previous research on scheduling problems in our framework considered either persistence of the tasks or dependency among the tasks but not both.

An early work by Liu and Layland [LL73] considers persistent scheduling of tasks in a single processor environment. In their problem, tasks have deadlines which specify a time limit in which the ith scheduling of a given task must have occurred. They give an algorithm which achieves full processor utilization for this task. More recently, the problem of scheduling persistent tasks has been studied in the work of Baruah et al. [BCPV96]. Their setting is closer to ours, and we describe their problem in detail. They considered the problem of scheduling a set of n tasks with given (arbitrary) frequencies on m machines. (The case $m = 1$ is equivalent to an instance of our problem in which the conflict graph is a clique.) To measure *regularity* of a schedule for their problem they introduced the notion of *P-fairness*. A schedule for this problem is P-fair (proportionate-fair) if at each time t for each task i the absolute value of the difference in the number of times i has been scheduled and $f_i t$ is strictly less than 1, where f_i is the frequency of task i. They provided an algorithm for computing a P-fair solution to their problem. Their problem fails to capture our situation for two reasons. First, we would like to constrain the sets of tasks that can be scheduled concurrently according to the topology of the conflict graph and not according to their cardinality. Moreover, in their problem every feasible frequency requirement can be scheduled in a P-fair manner. For our scheduling problem, we show that such a P-fair schedule cannot always be achieved. To deal with feasible frequencies that cannot be scheduled in a P-fair manner, we define weaker versions of regularity.

The dependency property captures most of the work done based on the well-known "dining philosophers" paradigm (see, for example, [Dijk71], [Lyn80], [CM84], [AS90], [CS92], and [BP92]). In this setting, Lynch [Lyn80] was the first to explicitly consider the *response time* for each task. The goal of successive works was to make the response time of a node depend only on its local neighborhood in the conflict graph (see, e.g., [BP92]). While response time in terms of a node's degree is adequate for "one-shot" tasks, it does not capture our requirement that a task should be scheduled in a regular and fair fashion over a period of time.

1.3. Notations and definitions. A *schedule* S is an infinite sequence of independent sets $I_1, I_2, \ldots, I_t, \ldots$. Let $S(i,t)$ denote the indicator variable that represents the schedule; it is 1 if task i is scheduled at time t and 0 otherwise. Let $f_i^{(t)}(S) = \sum_{\tau=1}^{t} S(i,\tau)/t$. We refer to $f_i^{(t)}(S)$ as the *prefix frequency* of task i at time t in schedule S. Let $f_i(S) = \liminf_{t\to\infty}\{f_i^{(t)}(S)\}$. We refer to $f_i(S)$ as the *frequency* of task i in schedule S. We say that a schedule S is *periodic* with *period* T, if $I_j = I_{T+j} = I_{2T+j} = \cdots$ for all $1 \le j \le T$. In periodic schedules we will refer to $f_i^{(T)}(S)$ as the frequency of task i. (In both $f_i^{(t)}(S)$ and $f_i(S)$ we drop the index S whenever the identity of the schedule S is clear from the context.)

DEFINITION 1. *A vector of frequencies* $\hat{f} = (f_1, \ldots, f_n)$ *is* feasible *if there exists a schedule* S *such that the frequency of the* i*th task in schedule* S *is at least* f_i.

DEFINITION 2. *A schedule* S realizes *a vector of frequencies* $\hat{f}$ *if the frequency of the* i*th task in schedule* S *is at least* f_i. *A schedule* S c-approximates *a vector of frequencies* $\hat{f}$ *if the frequency of the* i*th task in schedule* S *is at least* f_i/c.

A measure of fairness. Fairness is determined via a partial order $\prec$ that we define on the set of frequency vectors.

DEFINITION 3. *Given two frequency vectors* $\hat{f} = (f_1, \ldots, f_n)$ *and* $\hat{g} = (g_1, \ldots, g_n)$, $\hat{f} \prec \hat{g}$ *(*$\hat{f}$ *is less fair than* $\hat{g}$*) if there exists an index* i *and a threshold* f *such that* $f_i < f \le g_i$, *and for all* j *such that* $g_j \le f$, $f_j \le g_j$.

Less formally, if $\hat{f} \prec \hat{g}$, then $\hat{g}$ "performs" better for some task i and all tasks with frequency smaller than the ith task, i.e., those tasks that are least scheduled. It could be the case that $\hat{f}$ "performs" better for the other tasks; however, our concern in fair allocation is with the less frequently scheduled tasks.

In Appendix A we prove that the relation $\prec$ is indeed a partial order.

DEFINITION 4. *A vector of frequencies* $\hat{f}$ *is* max-min fair *if no feasible vector* $\hat{g}$ *satisfies* $\hat{f} \prec \hat{g}$.

Less formally, in a max-min fair frequency vector, one cannot increase the frequency of some task at the expense of less frequently scheduled tasks. This means that our goal is to let each task i have more of the resource as long as we have to take the resource away only from tasks which are better off, i.e., those that have more of the resource than task i.

Measures of regularity. We provide two measures by which one can evaluate a schedule for its *regularity*. We call these measures the *response time* and the *drift*.

Given a schedule S, the *response time* for task i, denoted r_i, is the largest interval of time for which task i waits between successive schedulings. More precisely,

$$r_i = \max\{t_2 - t_1 | 0 \le t_1 < t_2 \text{ s.t. } \forall_{t_1<t<t_2} S(i,t) = 0\}.$$

For any time t, the number of expected occurrences of task i can be expressed as $f_i t$. But note that if r_i is larger than $1/f_i$, it is possible that, for some period of time, a schedule allows a task to "drift away" from its expected number of occurrences. In

order to capture this, we introduce a second measure for the regularity of a schedule. We denote by d_i the *drift* of a task i. It indicates how much a schedule allows task i to drift away from its expected number of scheduled units:

$$d_i = \max_t \left\{ \left| f_i \cdot t - \sum_{r=1}^{t} S(i,r) \right| \right\}.$$

Note that if a schedule S achieves drift $d_i < 1$ for all i, then it is *P-fair* as defined in [BCPV96].

A schedule achieves its strongest form of regularity if each task i is scheduled every $1/f_i$ time units (except for its first appearance). We say that a schedule is *rigid* if for each task i there exists a starting point s_i such that the task is scheduled on exactly the time units $s_i + j(1/f_i)$ for $j \geq 0$.

Graph subclasses. A graph is *perfect* if for all its induced subgraphs the size of the maximum clique is equal to the chromatic number (cf. [Gol80]). A graph is an interval graph (circular-arc graph) if its vertices correspond to intervals on a line (circle), and two vertices are adjacent if the corresponding intervals intersect (cf. [Tuc71]).

1.4. Results. In section 2 we motivate our definition of max-min fairness and show several of its properties. First, we provide an equivalent (alternate) definition of feasibility which shows that deciding feasibility of a frequency vector is computable. Next, we prove that every graph has a *unique* max-min fair frequency vector. Then, we show that the task of even weakly approximating the max-min fair frequencies on general graphs is NP-hard. As we mentioned above, many practical applications of this problem arise from simpler networks such as buses and rings (i.e., interval conflict graphs and circular-arc conflict graphs). For the case of perfect graphs (and hence for interval graphs), we describe an efficient algorithm for computing max-min fair frequencies. We prove that the period T of a schedule realizing such frequencies on a perfect graph satisfies $T = 2^{O(n)}$ and that there exist interval graphs such that $T = 2^{\Omega(n)}$.

The rest of our results deal with the problem of finding the most "regular" schedule that realizes any feasible frequency vector. In section 3 we show the existence of interval graphs for which there is no P-fair schedule that realizes their max-min fair frequencies. In section 4 we describe an algorithm for computing a schedule that realizes any given feasible frequencies on interval graphs. The schedule computed by the algorithm achieves response time of $\lceil 4/f_i \rceil$ and drift of $O(\sqrt{\log T} n^{\epsilon})$. A slight modification of this algorithm yields a schedule that 2-approximates the given frequencies. The advantage of this schedule is that it achieves a bound of one on the drift and hence a bound of $\lceil 2/f_i \rceil$ on the response time. In section 5 we present an algorithm for computing a schedule that 12-approximates any given feasible frequencies on interval graphs and has the advantage of being rigid. All algorithms run in polynomial time. In section 6 we show how to transform any algorithm for computing a schedule that c-approximates any given feasible frequencies on interval graphs into an algorithm for computing a schedule that $2c$-approximates any given feasible frequencies on circular-arc graphs. The response time and drift of the resulting schedule are doubled as well.

Finally, in section 7 we summarize our results, list a number of open problems, and sketch what additional properties are required to obtain solutions for actual networks.

2. Max-min fair allocation. Our definition for max-min fair allocation is based on the definition used by Jaffe [Jaf81] and Bertsekas and Gallager [BG87] but differs

in one key ingredient, namely, our notion of feasibility. We study some elementary properties of our definition in this section. In particular, we show that our definition guarantees a unique max-min fair frequency vector for every conflict graph. We also show the hardness of computing the frequency vector for general graphs. However, for the special case of perfect graphs our notion turns out to be the same as that of [BG87].

The definition of [Jaf81] and [BG87] is considered as the traditional way of measuring throughput fairness in communication networks and is also based on the partial order $\prec$ as used in our definition. The primary difference between our definition and theirs is in the definition of *feasibility*. Bertsekas and Gallager [BG87] use a definition, which we call clique feasibility, that is defined as follows:

> a vector of frequencies $(f_1, \ldots, f_n)$ is *clique feasible* for a conflict graph G, if $\sum_{i \in C} f_i \leq 1$ for all cliques C in the graph G.

The notion of max-min fairness of Bertsekas and Gallager [BG87] is now exactly our notion, with feasibility replaced by clique feasibility.

The definition of [BG87] is useful for capturing the notion of fractional allocation of a resource such as bandwidth in a communication networks. However, in our application we need to capture a notion of integral allocation of resources, and hence their definition does not suffice for our purposes. By definition, every frequency vector that is feasible in our sense is clique feasible. However, the converse is not true. Consider the five-cycle conflict graph. For this graph the vector $(1/2, 1/2, 1/2, 1/2, 1/2)$ is clique feasible, but no schedule can realize this frequency vector.

2.1. An alternate definition of feasibility. Given a conflict graph G, let $\mathcal{I}$ denote the family of all independent sets in G. For $I \in \mathcal{I}$, let $\chi(I)$ denote the characteristic vector of I.

THEOREM 5. *A vector of frequencies $\hat{f}$ is feasible if and only if $\hat{f}$ is a convex combination of the $\chi(I)$'s; that is, there exist weights $\{\alpha_I\}_{I \in \mathcal{I}}$ such that $\sum_{I \in \mathcal{I}} \alpha_I = 1$ and $\sum_{I \in \mathcal{I}} \alpha_I \chi(I) = \hat{f}$.*

Proof. Claim 6 proves the easier direction. Claim 7 proves the other direction only for the case when $\hat{f}$ can be expressed as a rational convex combination of the independent sets. The proof for the case when $\hat{f}$ is not a rational convex combination is given in Appendix B. □

CLAIM 6. *If a frequency vector $\hat{f}$ is feasible then there exists a sequence of weights $\{\alpha_I\}_{I \in \mathcal{I}}$ such that $\sum_I \alpha_I = 1$ and $\sum_I \alpha_I \chi(I) = \hat{f}$.*

Proof. To obtain a contradiction assume otherwise. By continuity there exists an $\epsilon > 0$, such that the vector $(1 - \epsilon)\hat{f}$ cannot be expressed as a convex combination of the $\chi(I)$'s. Based on the definition of feasibility, there exists a schedule S which achieves a frequency of at least $\hat{f}$. In particular, there exists a time $T = T_\epsilon$ such that $f_i^{(T)} \geq f_i - \epsilon$ for all $1 \leq i \leq n$. Let α_I be the frequency of the independent set I in the first T time units in the schedule S. Then $\sum_I \alpha_I = 1$ and $\sum_I \alpha_I \chi(I) \geq (1 - \epsilon)\hat{f}$, contradicting the choice of ϵ. □

CLAIM 7. *If a frequency vector $\hat{f}$ can be expressed as a rational convex combination of the independent sets, then $\hat{f}$ is feasible.*

Proof. Suppose that there exist rational weights $\{\alpha_I\}_{I \in \mathcal{I}}$, such that $\sum_{I \in \mathcal{I}} \alpha_I = 1$ and $\sum_{I \in \mathcal{I}} \alpha_I \chi(I) = \hat{f}$. Express α_I as p_I/q_I where p_I and q_I are integral and let $T = \text{LCM}\{q_I\}$. Observe that $N_I \equiv \alpha_I T$ is integral. For each I we schedule N_I times the independent set I over a period of T intervals (in any arbitrary N_I units of time). It is clear that there are enough slots for each independent set to be scheduled N_I times. □

The main impact of Theorem 5 is that it shows that the space of all feasible frequencies is well behaved (i.e., it is a closed, connected, compact space). In addition, it shows that determining whether a frequency vector is feasible is a computable task (a fact that may not have been easy to see from the earlier definition). We now use this definition to observe the following interesting connection.

PROPOSITION 8. *Given a conflict graph G, the notions of feasibility and clique feasibility are equivalent if and only if G is perfect.*

Proof. The proof follows directly from well-known polyhedral properties of perfect graphs. (See [GLS87], [Knu94].) In the notation of Knuth [Knu94] the space of all feasible vectors is the polytope STAB(G), and the space of all clique-feasible vectors is the polytope QSTAB(G). The result follows from the theorem on page 38 in [Knu94] which says that a graph G is perfect if and only if STAB(G) = QSTAB(G). □

2.2. Uniqueness and computability of max-min fair frequencies. In this subsection we prove that the max-min fair frequency vector is unique. We also show that finding this vector (or even approximating it) is computationally hard.

THEOREM 9. *For any conflict graph there exists a unique max-min fair frequency vector.*

Proof. For a vector $\hat{f}$ let sort$\hat{f}$ denote the vector obtained by permuting the vector $\hat{f}$ so that its coordinates appear in nondecreasing order. Let the relation $\prec_{\text{lex}}$ on the feasible frequency vectors be the lexicographic ordering on sort$\hat{f}$. More precisely, $\hat{f} \prec_{\text{lex}} \hat{g}$ if either sort$\hat{f}$ = sort$\hat{g}$ or there exists an index $i \geq 1$ such that sort$\hat{f}_i <$ sort$\hat{g}_i$ and sort$\hat{f}_j$ = sort$\hat{g}_j$ for all $1 \leq j < i$. (Note that $\prec_{\text{lex}}$ is not a partial order since it is not antisymmetric.) It is easy to verify that if $\hat{f} \prec_{\text{lex}} \hat{g}$, and sort$\hat{f} \neq$ sort$\hat{g}$ then $f \prec g$. Let the vector $\hat{f}$ be a max-min fair vector. (Such a vector exists in the space of feasible vectors, since this space is compact.) The vector $\hat{f}$ is also larger according to the ordering $\prec_{\text{lex}}$ than any vector $\hat{g}$ such that sort$\hat{f} \neq$ sort$\hat{g}$. We now show that there is no vector $\hat{g}$ such that sort$\hat{f}$ = sort$\hat{g}$. This implies that it is the unique max-min fair frequency vector.

To obtain a contradiction, suppose that there exists a vector $\hat{g}$ such that sort$\hat{f}$ = sort$\hat{g}$. First observe that the vector $\hat{h} = (\hat{f} + \hat{g})/2$ is feasible. This is true because $\hat{f}$ and $\hat{g}$ can be expressed as a convex combination of the independent sets and $\hat{h}$ is a convex combination of $\hat{f}$ and $\hat{g}$. Thus $\hat{h}$ is a convex combination of the independent sets. Now assume without loss of generality that the indices of the vectors are arranged in increasing order of $f_i + g_i$. Let j be the smallest index such that $f_j \neq g_j$. Say f_j is the smaller of the two. Then $((\hat{f} + \hat{g})/2)_j$ is greater than f_j and for all vertices i with smaller frequencies, $f_i = ((\hat{f} + \hat{g})/2)_i$. This implies that $\hat{f} \prec_{\text{lex}} (\hat{f} + \hat{g})/2$, which is a contradiction. □

We now turn to the issue of the computability of the max-min fair frequencies. While we do not know the exact complexity of computing max-min fair frequencies (in particular, we do not know if deciding whether a frequency vector is feasible is in NP ∪ coNP), it does seem to be very hard in general. Here, we consider the subproblem of computing the smallest frequency assigned to any vertex by a max-min allocation and show the following theorem.

THEOREM 10. *There exists an $\epsilon > 0$ such that given a conflict graph on n vertices, approximating the smallest frequency assigned to any vertex in a max-min fair allocation to within a factor of n^{ϵ} is NP-hard.*

Proof. We relate the computation of max-min fair frequencies in a general graph to the computation of the fractional chromatic number of a graph. We then use the

recent hardness result for approximating the (fractional) chromatic number due to Lund and Yannakakis [LY93] to show that computing max-min fair frequencies in general graphs is very hard.

The fractional chromatic number problem (cf. [LY93]) is defined as follows.

> To each independent set I in the graph, assign a weight w_I, so as to minimize the quantity $\sum_I w_I$, subject to the constraint that for every vertex v in the graph the quantity $\sum_{I \ni v} w_I$ is at least 1. The quantity $\sum_I w_I$ is called the *fractional chromatic number* of the graph.

Observe that if the w_I's are forced to be integral, then the fractional chromatic number is the chromatic number of the graph.

The following claim shows a relationship between the fractional chromatic number and the assignment of feasible max-min fair frequencies.

CLAIM 11. *Let $(f_1, f_2, \ldots, f_n)$ be a feasible assignment of frequencies to the vertices in a graph G. Then $1/(\min_i f_i)$ is an upper bound on the fractional chromatic number of the graph. Conversely, if k is the fractional chromatic number of a graph, then a schedule that sets the frequency of every vertex to be $1/k$ is feasible.*

The proof of the above claim is straightforward given the definitions of fractional chromatic number and feasibility. We now show how to use the claim to prove the theorem.

The above claim, combined with the hardness of computing the fractional chromatic number [LY93], suffices to show the NP-hardness of deciding whether a given assignment of frequencies is feasible for a given graph. To show that the claim also implies the hardness of approximating the smallest frequency in the max-min fair frequency vector we inspect the Lund–Yannakakis construction a bit more closely. Their construction yields a graph in which every vertex participates in a clique of size k such that deciding if the (fractional) chromatic number is k or kn^ϵ is NP-hard. In the former case, the max-min fair frequency assignment to every vertex is at least $1/k$. In the latter case at least some vertex will have frequency smaller that $1/(kn^\epsilon)$. Thus this implies that approximating the smallest frequency in the max-min fair frequencies to within a factor of n^ϵ is NP-hard. □

2.3. Max-min fair frequencies on perfect graphs. We now consider perfect graphs. We show how to compute in polynomial time max-min fair frequencies for this class of graphs and give bounds on the period of a schedule realizing such frequencies. As our main focus of the subsequent sections will be interval graphs, we will give our algorithms and bounds first in terms of this subclass and then show how to generalize the results to perfect graphs.

We start by describing an algorithm for computing max-min fair frequencies on interval graphs. As we know that clique feasibility equals feasibility (by Proposition 8), we can use an adaptation of [BG87].

ALGORITHM 1. *Let $\mathcal{C}$ be the collection of maximal cliques in the interval graph. (Notice that $\mathcal{C}$ has at most n elements and can be computed in polynomial time.) For each clique $C \in \mathcal{C}$ the algorithm maintains a* residual capacity *which is initially* 1. *To each vertex the algorithm associates a label* assigned/unassigned. *All vertices are initially unassigned. Dividing the residual capacity of a clique by the number of unassigned vertices in this clique yields the* relative *residual capacity. Iteratively, we consider the clique with the smallest current relative residual capacity and assign to each of the clique's unassigned vertices this capacity as its frequency. For each such vertex in the clique we mark it assigned and subtract its frequency from the residual*

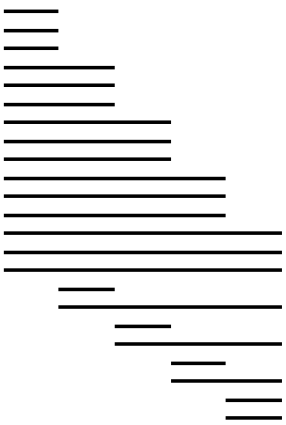


FIG. 1. *An interval graph with $n = 23$ intervals for which $T = LCM_{i=1}^{n}\{q_i\} \geq 2^{\frac{n+2}{5}-1}$.*

capacity of every clique that contains it. We repeat the process till every vertex has been assigned some frequency.

It is not hard to see that Algorithm 1 correctly computes max-min fair frequencies in polynomial time. We now use its behavior to prove a tight bound on the period of a schedule for an interval graph. The following theorem establishes this bound.

THEOREM 12. *Let $f_i = p_i/q_i$ be the frequencies in a max-min fair schedule for an interval graph G, where p_i and q_i are relatively prime. Then, the period for the schedule $T = LCM_{i=1}^{n}\{q_i\}$ satisfies, $T = 2^{O(n)}$. Furthermore, there exist interval graphs for which $T = 2^{\Omega(n)}$.*

We prove this theorem with the help of the following two lemmas.

LEMMA 13. *$LCM_{i=1}^{n}\{q_i\} \leq 2^{n/2}$.*

Proof. Let n_j denote the number of intervals that are assigned frequency $\frac{p_j}{q_j}$ in iteration j. That is, $\frac{p_j}{q_j}$ is the minimum relative residual capacity at iteration j. From the way the relative residual capacities are updated, it follows that q_i divides $\prod_{j=1}^{i} n_j$ for all $1 \leq i \leq n$. The lemma follows since assuming there were ℓ iterations, q_i divides $\prod_{j=1}^{\ell} n_j$, and $\prod_{j=1}^{\ell} n_j$ attains its maximum when $\ell = n/2$ and $n_i = 2$ for all $1 \leq i \leq \ell$. □

LEMMA 14. *There exists an interval graph for which $LCM_{i=1}^{n}\{q_i\} \geq 2^{\frac{n+2}{5}-1}$.*

Proof. We show an interval graph in which $\max_{i=1}^{n}\{q_i\} \geq 2^{\frac{n+2}{5}-1}$; the lemma follows since trivially $\mathrm{LCM}_{i=1}^{n}\{q_i\} \geq \max_{i=1}^{n}\{q_i\}$. For simplicity we assume that 5 divides $n + 2$. The reader may find it easier to follow the construction for $n = 23$ depicted in Figure 1. Let $x = \frac{3(n+2)}{5}$ and $y = \frac{n+2}{5} - 1$. In the construction x intervals start at 0 (the top 15 intervals in Figure 1), and two intervals start at i for all $1 \leq i \leq y$ (the bottom 8 intervals in Figure 1). Note that indeed $2y + x = n$. Out of the first x intervals, three intervals terminate at i for each $1 \leq i \leq y + 1$. Note that indeed $3(y + 1) = x$. For $1 \leq i \leq y$, out of the two intervals starting at i, one interval terminates at $i + 1$ and one continues until $y + 1$.

Now, since at 0 the size of the clique is x and at i the size of the clique is $x - 3i + (i - 1) + 2 = x - 2i + 1$, it follows that the algorithm for the frequency assignment handles the cliques from left to right and there are $y + 1$ different values for frequencies denoted by $w_0, \ldots, w_y$. We get

$$w_0 = \frac{1}{x},$$
$$w_1 = \frac{3w_0}{2} = \frac{3}{2x},$$
$$w_2 = \frac{3w_0 + w_1}{2} = \frac{9}{4x}.$$

In general, by induction we prove that $w_i = \frac{3}{x} \cdot \frac{2^i - 1}{2^i}$ for $1 \le i \le y$.

$$w_{i+1} = \frac{3w_0 + w_i}{2} = \frac{3}{2x} + \frac{3}{x} \cdot \frac{2^i - 1}{2^{i+1}} = \frac{3}{x} \cdot \frac{1}{2} + \frac{2^i - 1}{2^{i+1}} = \frac{3}{x} \cdot \frac{2^{i+1} - 1}{2^{i+1}}.$$

Since the denominator of w_y is greater than or equal to 2^y and neither 3 nor $2^y - 1$ has a common divisor with 2^y, it follows that $\max_{i=1}^{n}\{q_i\} \ge 2^{\frac{n+2}{5}-1}$. □

Algorithm 1 works for all graphs where clique feasibility determines feasibility, i.e., perfect graphs. However, the algorithm does not remain computationally efficient since it involves scanning all the cliques in the graph. Still, Theorem 12 can be directly extended to the class of perfect graphs. We now use this fact to describe a polynomial-time algorithm for assigning max-min fair frequencies to perfect graphs.

ALGORITHM 2. *This algorithm maintains the labelling procedure* assigned/unassigned *of Algorithm* 1. *At each phase, the algorithm starts with a set of assigned frequencies and tries to find the largest f such that all unassigned vertices can be assigned the frequency f. To compute f in polynomial time, the algorithm uses the fact that deciding if a given set of frequencies is feasible is reducible to the task of computing the size of the largest weighted clique in a graph with weights on vertices. The latter task is well known to be computable in polynomial time for perfect graphs. Using this decision procedure the algorithm performs a binary search to find the largest achievable f. (The binary search does not have to be too refined due to the upper bound on the denominators of the frequencies given in Theorem* 12.*) Having found the largest f, the algorithm finds a set of vertices which are saturated under f as follows: let ϵ be some small number, with the property that the difference between any two distinct assigned frequencies is more than ϵ. By Theorem* 12, $\epsilon = 2^{-n^2}$ *is sufficient. Now the algorithm raises, one at a time, the frequency of each unassigned vertex to $f + \epsilon$, while maintaining the other unassigned frequencies at f. If the so obtained set of frequencies is not feasible, then it marks the vertex as assigned and its frequency is assigned to be f. The algorithm now repeats the phase until all vertices have been assigned some frequency.*

3. Nonexistence of *P*-fair allocations. We show that a P-fair scheduling under max-min fair frequencies need not exist for every interval graph.

THEOREM 15. *There exist interval graphs G for which there is no P-fair schedule that realizes their max-min frequency assignment.*

Proof. In order to prove the theorem we produce a counterexample as follows. We choose a parameter k and for every permutation π of the elements $\{1, \ldots, k\}$, we define an interval graph G_π. We show a necessary condition that π must satisfy if G_π has a P-fair schedule. Last, we show that there exists a permutation π of 12 elements which does not satisfy this condition.

Given a permutation π on k elements, G_π consists of $3k$ intervals. For $1 \le 1 \le k$, define the intervals $A(i) = (i-1, i]$, $B(i) = (i, k+\pi(i)+1]$ and $C(i) = (k+i+1, k+i+2]$. Observe that the max-min frequency assignment to G_π is the following: all the tasks $B(1), \ldots, B(k)$ have frequency $1/k$; task $A(i)$ has frequency $(k-i+1)/k$ for $1 \le i \le k$; and task $C(i)$ has frequency i/k for $1 \le i \le k$. (See Figure 2.)

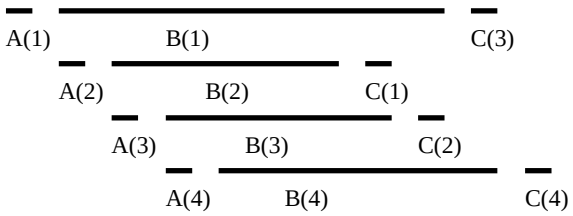


FIG. 2. *The graph G_π for $\pi = (3, 1, 2, 4)$.*

We now observe the properties of a P-fair schedule for the tasks in G_π. (i) The time period is k. (ii) The schedule is entirely specified by the schedule for the tasks $B(i)$. (iii) This schedule is a permutation σ of k elements, where $\sigma(i)$ is the time unit for which $B(i)$ is scheduled. To see what kind of permutations σ constitute P-fair schedules of G_π we define the notion of when a permutation is *fair* for another permutation.

DEFINITION 16. *A permutation σ_1 is* fair *for a permutation σ_2 if for all $1 \leq i, j \leq k$, σ_1 and σ_2 satisfy the conditions* cond_{ij} *defined as follows:*

$$\frac{ij}{k} - 1 < |\{\ell : \sigma_1(\ell) \leq j \text{ and } \sigma_2(\ell) \leq i\}| < \frac{ij}{k} + 1 \ .$$

CLAIM 17. *If a permutation σ is a P-fair schedule for G_π then σ is fair for the identity permutation and permutation π.*

Proof. For fixed i, we claim that the conditions cond_{ij} for $\sigma_1 = \sigma$ and σ_2 being the identity permutation are exactly the conditions for a P-fair allocation of $A(i+1)$. Similarly, the conditions cond_{ij} for $\sigma_1 = \sigma$ and $\sigma_2 = \pi$ are the conditions for a P-fair allocation of $C(k-i)$. Thus, a permutation σ represents a P-fair schedule for G_π if and only if σ is fair for both π and the identity permutation.

We now show why the conditions cond_{ij} for $\sigma_1 = \sigma$ and σ_2 being the identity permutation are exactly the conditions for a P-fair allocation of $A(i+1)$. The claim about the conditions cond_{ij} for $\sigma_1 = \sigma$ and $\sigma_2 = \pi$ is analogous. Recall that the frequency of task $A(i+1)$ is $(k-i)/k$ and that $A(i+1)$ can be scheduled only when tasks $B(\ell)$, for $1 \leq \ell \leq i$, are not scheduled. Consider the schedule up to time $j \leq k$. In order for the schedule to be P-fair, the number of occurrences of tasks $B(\ell)$, for $1 \leq \ell \leq \ell$, up to this time must be between $j - \frac{(k-i)j}{k} - 1 = \frac{ij}{k} - 1$ and $j - \frac{(k-i)j}{k} + 1 = \frac{ij}{k} + 1$. Note that the number of times these tasks are scheduled is the cardinality of the set $\{\ell : \sigma_1(\ell) \leq j \text{ and } \ell \leq i\}$, which translates to cond_{ij} for $\sigma_1 = \sigma$ and σ_2 being the identity permutation. □

Let $\pi = (1, 3, 4, 7, 8, 9, 11, 5, 12, 10, 2, 6)$ be a permutation on 12 elements. The following arguments show that no permutation σ is fair to both the identity permutation and the permutation π.

1. We define a *block* as any contiguous set of elements in the range 1 to 12. We say that σ places an element i in the block $[j, \ell]$.
2. Without loss of generality assume that σ places the element 1 in the block $[1, 6]$.
3. Consider the elements in the following six pairs $\{1, 2\}$, $\{3, 4\}$, $\{5, 6\}$, $\{7, 8\}$, $\{9, 10\}$, and $\{11, 12\}$. If the permutation σ is fair for the identity permutation, then it must place exactly one element of each pair in the block $[1, 6]$. To see this, note that if σ places both elements 1 and 2 in the block $[1, 6]$, then $|\{\ell : \sigma(\ell) \leq 6 \text{ and } \ell \leq 2\}| = 2$, violating $\text{cond}_{2,6}$. Thus only element 1 is in

block $[1,6]$. Inductively, it can be shown that if σ places both elements $2i-1$ and $2i$ for $1 < i \leq 6$ in the block $[1,6]$, then $\text{cond}_{2i,6}$ is violated.

4. A similar argument applied to π implies that σ must place exactly one element of each of the six pairs $\{1,3\}$, $\{4,7\}$, $\{8,9\}$, $\{11,5\}$, $\{12,10\}$, and $\{2,6\}$ in the block $[1,6]$ if σ is fair for π.
5. Arguments 2, 3, and 4 imply that the first half of σ consists of the elements $\{1,4,8,10,11,6\}$ and the second half consists of the elements $\{3,7,9,12,5,2\}$.
6. Again, since σ is fair for the identity permutation, σ must place exactly one of the elements of each of the triplets $\{1,2,3\}$, $\{4,5,6\}$, and $\{7,8,9\}$ in each of the blocks $[1,4]$, $[5,8]$, and $[9,12]$ in order not to violate conditions $\text{cond}_{3,4}$, $\text{cond}_{3,8}$, and $\text{cond}_{3,12}$.
7. Similarly, since σ is fair for π, σ must place exactly one of the elements of the triplet $\{\pi(7),\pi(8),\pi(9)\} = \{11,5,12\}$ in each of the blocks $[1,4]$, $[5,8]$, and $[9,12]$.
8. Since 1 appears in the block $[1,6]$ and both 2 and 3 appear in the block $[7,12]$, it follows from Argument 6 that exactly one of the elements 2 and 3 is placed in the block $[7,8]$ by σ.
9. A similar argument applied to the triplet $\{7,8,9\}$ implies that exactly one of 7 and 9 is placed in the block $[7,8]$ by σ.
10. Last, we examine the triplet $\{\pi(7),\pi(8),\pi(9)\} = \{11,5,12\}$. It follows from Argument 7 that one of 5 and 12 must appear in the block $[7,8]$.

Since σ cannot place three elements in a block of size two, we obtain the contradiction.

The proof of Theorem 15 follows. □

4. Realizing frequencies exactly. In this section we first show how to construct a schedule which realizes any feasible set of frequencies (and hence in particular max-min frequencies) exactly on an interval graph. We prove its correctness and give a bound of $\lceil 4/f_i \rceil$ on the response time for each interval i. We then proceed to introduce a potential function which can be used to yield a bound of $O(n^{\frac{1}{2}+\epsilon})$ on the drift for every interval. An easy consequence of our algorithm is for the special case in which the frequencies are of the form $1/2^i$, the drift can be bounded by 1 and thus the waiting time can be bounded by $\lceil 2/f_i \rceil$. This yields a 2-approximation algorithm with high regularity.

Input to the algorithm a unit of time t and a conflict graph G which is an interval graph. The graph G is represented by a set $I = \{I_1, \ldots, I_n\}$ of intervals on the unit interval $[0,1]$ of the x-coordinate, where $I_i = [i.s, i.e]$ for $1 \leq i \leq n$. Every interval I_i has a frequency $f_i = p_i/q_i$ with the following constraint: $\sum_{I_i \ni x} f_i \leq 1$ for all $0 \leq x \leq 1$. For simplicity, we assume from now on that these constraints on the frequencies are met with equality and that $t \leq T = \text{LCM}\{q_i\}$.

Output of the algorithm an independent set I_t defining the set of tasks scheduled for time t such that the scheduled S, given by $\{I_t\}_{t=1}^{T}$ realizes frequencies f_i.

The algorithm is recursive. Let s_i denote the number of times a task i has to appear in T time units, i.e., $s_i = Tp_i/q_i$. The algorithm has $\log T$ levels of recursion. In the first level we decide on the occurrences of the tasks in each half of the period. That is, for each task we decide how many of its occurrences appear in the first half of the period and how many in the second half. This yields a problem of a recursive nature in the two halves. In order to find the schedule at time t, it suffices to solve the problem recursively in the half which contains t. (Note that in case T is odd one of the halves is longer than the other.) Clearly, if a task has an even number of occurrences in T it would appear the same number of times in each half in order to minimize the

drift. The problem is with tasks that have an odd number of occurrences s_i. Clearly, each half should have at least $\lfloor s_i \rfloor$ of the occurrences. The additional occurrence has to be assigned to one of the halves in a way that both resulting subproblems would still be feasible. This is the main difficulty of the assignment and is solved in the procedure *sweep*.

Procedure sweep. In this procedure we compute the assignment of the additional occurrence for all tasks that have an odd number of occurrences. The input to this procedure is a set of intervals $I_1, \ldots, I_m$ (those having odd s_i's) with the restriction that each clique in the resulting interval subgraph is of even size. (Later, we show how to overcome this restriction.) The output is a partition of these intervals into two sets such that each clique is equally divided among the sets. This is done by a sweep along the x-coordinate of the intervals. During the sweep every interval will be assigned a variable which at the end is set to 0 or 1 (i.e., first half of the period or second half of the period). Suppose that we sweep point x. We say that an interval I_i is *active* while we sweep point x if $x \in I_i$. The assignment rules are stated in Algorithm 3.

ALGORITHM 3. *For each interval I_i that starts at x:*
If the current number of active intervals is even:
A new variable is assigned to I_i (I_i is unpaired*).*
Otherwise; the current number of active intervals is odd:
I_i is paired to the currently unpaired interval I_j, and it is assigned the negation of I_j's variable.
Comment: *No matter what value is later assigned to this variable, I_i and I_j will end up in opposite halves.*

ALGORITHM 4. *For each interval I_i that ends at x:*
If the current number of active intervals is even:
Nothing is done.
Otherwise; the current number of active intervals is odd:
If I_i is paired with I_j:
I_j is now paired with the currently unpaired interval I_k. Also, I_j's variable is matched with the negation of I_k's variable.
Comment: *This will ensure that I_j and I_k are put in opposite halves, or equivalently, I_i and I_k are put in the same halves.*
If I_i is unpaired:
Assign arbitrarily 0 *or* 1 *to I_i's variable.*

It will be proven later that these rules ensure that whenever the number of active intervals is even, then exactly half of the intervals will be assigned 0 and half will be assigned 1. We note that since the conflict graph is an interval graph we are assured that when we apply the above rules pairing up arbitrary intervals will not result in a circular dependency of the variables (e.g., $x = y = \bar{x}$).

Recall that we assumed that the size of each clique is even. To overcome this restriction we need the following simple lemma. For $x \in [0, 1]$, denote by C_x the set of all the input intervals (with odd and even s_i's) that contain x; C_x will be referred to as a clique.

LEMMA 18. *The period T is even if and only if $|\{i \ : \ I_i \in C \wedge s_i \text{ is odd}\}|$ is even for every clique C.*

Proof. Note that

$$\sum_{I_i \in C} f_i = 1 \Rightarrow \sum_{I_i \in C} s_i = T \Rightarrow \sum_{I_i \in C,\ s_i \text{ is odd}} s_i + \sum_{I_i \in C,\ s_i \text{ is even}} s_i = T.$$

Since the second summand is always even, T is even if and only if the first summand is also even. □

This lemma implies that if T is even then the size of each clique in the input to procedure *Sweep* is indeed even. If T is odd, then a dummy interval I_{n+1} which extends over all other intervals and which has exactly one occurrence is added to the set I before calling *sweep*. Again, by Lemma 18, we are sure that in this modified set I the size of each clique is even. This would increase the period by one. The additional time unit will be allotted only to the dummy interval and thus can be ignored. We note that to produce the schedule at time t we just have to follow the recursive calls that include t in their period.

Applying this algorithm to the max-min frequencies yields a polynomial in n algorithm. This is true because there are no more than $\log T$ such calls and because $T = 2^{O(n)}$ for max-min fair frequencies.

LEMMA 19. *The algorithm produces a correct schedule for every feasible set of frequencies.*

Proof. We need to prove that feasibility is maintained with every recursive step. We show that the following invariant is maintained by *sweep.*

> For every x for which the number of active intervals in C_x is even, exactly half of the intervals will be assigned 0 and half will be assigned 1.

This invariant is easily maintained when a new interval starts: if the current number of intervals is odd, then the new interval is paired up with the currently unpaired interval and thus will be scheduled in the opposite half of its partner. The invariant holds also when an interval ends since by our rules whenever an interval ends any two unpaired intervals are immediately paired up.

Now, if T is odd, then a dummy interval is added and hence *sweep* produces a feasible solution for $T + 1$. In this case the algorithm assigns the "smaller" half of T to the half to which *sweep* assigned the dummy interval and feasibility is maintained. □

LEMMA 20. *If the set of frequencies is of the form $1/2^i$ then the resulting schedule is P-fair (i.e., the drift can be bounded by 1) and the response time is bounded by $\lceil 2/f_i \rceil$.*

Proof. Since our algorithm always divides even s_i into equal halves, the following invariant is maintained: in recursion level j, if $s_i > 1$ then s_i is even. Also note that $T = 2^k$, where $\min_i f_i = 1/2^k$ and thus each s_i is of the form $2^{\psi_i - k}$. Now, following the algorithm, it can be easily shown that there is at least one occurrence of task i in each time interval of size $2^{k-\psi_i}$. This implies that $\lfloor \frac{t}{2^{k-\psi_i}} \rfloor \leq \sum_{r=1}^{t} S(i,r) \leq \lceil \frac{t}{2^{k-\psi_i}} \rceil$ and thus P-fairness follows. Since the drift is bounded by one the response time is bounded by $\lceil 2/f_i \rceil$. □

LEMMA 21. *The response time for every interval I_i is bounded by $\lceil 4/f_i \rceil$.*

Proof. Lemma 20 implies the case in which the frequencies are powers of two. Moreover, in case the frequencies are not powers of two, we can virtually partition each task into two tasks with frequencies a_i and b_i, respectively, so that $f_i = a_i + b_i$, a_i is a power of two, and $b_i < a_i$. Then, the schedule of the task with frequency a_i has drift 1. This implies that its response time is at most $\lceil 2/a_i \rceil \leq \lceil 4/f_i \rceil$. □

Remark. It can be shown that the bound of the above lemma is tight for our algorithm.

We summarize the results in this section in the following theorem.

THEOREM 22. *Given an arbitrary interval graph as a conflict graph, the algorithm exactly realizes any feasible frequency vector and guarantees that the response time is at most* $\lceil 4/f_i \rceil$.

4.1. Bounding the drift. Since the algorithm has $O(\log T)$ levels of recursion and each level may increase the drift by one, it follows that the maximum drift is bounded by $O(\log T)$. In this section we prove that we can decrease the maximum drift to be $O(\sqrt{\log T} n^{\epsilon})$ for any fixed ϵ, where n is the number of tasks. By Lemma 13 this implies that in the worst case the drift for max-min fair frequencies is bounded by $O(n^{\frac{1}{2}+\epsilon})$.

Our method to get a better drift is based on the following observation: at each recursive step of the algorithm two sets of tasks are produced such that each set has to be placed in a different half of the time interval currently considered. However, we are free to choose which set goes to which half. We use this degree of freedom to decrease the bound on the drift. To make the presentation clearer we assume that T is a power of two and that the time units are $0, \ldots, T-1$. The arguments can be modified to hold in the general case.

Consider a subinterval of size $T/2^j$ starting *after* time $t_\ell = i \cdot T/2^j - 1$ and ending at $t_r = (i+1) \cdot T/2^j - 1$ for $0 \le i \le 2^j - 1$. In the first j recursion levels we already fixed the number of occurrences of each task up to t_ℓ. Given this number, the drift d_ℓ at time t_ℓ is fixed. Similarly, the drift d_r at time t_r is also fixed. At the next recursion level we split the occurrences assigned to the interval $[t_\ell + 1, t_r]$, thus fixing the drift d_m at time $t_m = (t_\ell + t_r)/2$. Optimally, we would like the drifts after the next recursion level at each time unit $t \in [t_\ell + 1, t_r]$ to be the weighted average of the drifts d_ℓ and d_r. In other words, let $\alpha = (t - t_\ell)/(t_r - t_\ell)$; then we would like the drift at time t to be $\alpha d_r + (1-\alpha) d_\ell$. In particular, we would like the drift at t_m to be $(d_\ell + d_r)/2$. This drift can be achieved for t_m only if the occurrences in the interval $[t_\ell + 1, t_r]$ can be split equally. However, in case we have an odd number of occurrences to split, the drift at t_m is $(d_\ell + d_r)/2 \pm 1/2$, depending on our decision in which half interval to put the extra occurrence. Note that the weighted average of the drifts of all other points changes accordingly. That is, if the new d_m is $(d_\ell + d_r)/2 + x$, for $x \in \{\pm 1/2\}$, then the weighted average in $t \in [t_\ell + 1, (t_r + t_\ell)/2]$ is $\alpha d_r + (1-\alpha) d_\ell + 2\alpha x$, where $\alpha = (t - t_\ell)/(t_r - t_\ell) \le 1/2$, and the weighted average in $t \in [(t_r + t_\ell)/2 + 1, t_r]$ is $\alpha d_r + (1-\alpha) d_\ell + 2(1-\alpha)x$, where $\alpha = (t - t_\ell)/(t_r - t_\ell) > 1/2$.

Consider now the two sets of tasks S_1 and S_2 that we have to assign to the two subintervals (of the same size) at level k of the recursion. For each of the possible two assignments, we compute a "potential" based on the resulting drifts at time t_m. For a given possibility let $D[t_m, i, k]$ denote the resulting drift of task i at t_m after k recursion levels. Define the potential of t_m after k levels as $POT(t_m, k) = \sum_{i=1}^{n} D(t_m, i, j)^{\phi}$ for some fixed even constant ϕ. We choose the possibility with the lowest potential. We now prove that using this policy the drift of any task after $\log T$ steps is bounded by $O(\sqrt{\log T} \cdot n^{\frac{1}{\phi}})$.

Consider a time t and a task i. The drift of task i at t is the outcome of at most $\log T$ recursion levels. Define the drift of task i at t after k levels, denoted $D(t, i, k)$, as the weighted average drift at t given the fixed drifts after k levels. It is easy to see that the initial drift is zero, and the final weighted average drift is the actual drift at t. Also, in each level the drift may either stay the same (in case we have to split an even number of occurrences of task i) or is changed by $\pm x$ where $0 \le x \le 1/2$. Note that x is positive if and only if the change in the drift at the current median point closest to t is $+1/2$. We extend the definition of potentials to all time points t in the

obvious way; that is, $POT(t,k) = \sum_{i=1}^{n} D(t,i,k)^{\phi}$. We show that the potential after $\log T$ levels is bounded by $O(T^{\phi/2} \cdot n)$. This implies the desired bound on the drift of each task at t since the potential is the sum of the drifts to the power of ϕ.

LEMMA 23. *For all* $0 \le t \le T-1$, *and all* $1 \le k \le \log T$,

$$POT(t,k) \le POT(t,k-1) + c \cdot \sum_{i=1}^{n} D(t,i,k-1)^{\phi-2}$$

for some constant c.

Proof. The increment of the potential at time t at the kth level is bounded by the maximum over all disjoint sets $S_1, S_2 \subset \{1,\dots,n\}$ such that $|S_1| = |S_2|$ of

$$\begin{aligned}\min_{S_1,S_2} \Bigg\{ & \sum_{i\in S_1} [D(t,i,k-1)+x]^{\phi} + \sum_{i\in S_2} [D(t,i,k-1)-x]^{\phi}, \\ & \sum_{i\in S_2} [D(t,i,k-1)+x]^{\phi} + \sum_{i\in S_1} [D(t,i,k-1)-x]^{\phi} \Bigg\} \\ - & \sum_{i\in S_1\cup S_2} [D(t,i,k-1)]^{\phi}\end{aligned}$$

for some $0 \le x \le 1/2$. Since the minimum is always bounded by the average, the change is bounded by

$$\frac{1}{2}\left\{ \sum_{i\in S_1\cup S_2} [D(t,i,k-1)+x]^{\phi} + [D(t,i,k-1)-x]^{\phi} - 2[D(t,i,k-1)]^{\phi} \right\}.$$

Finally, the maximum over all disjoint sets $S_1, S_2 \subset \{1,\dots,n\}$ such that $|S_1| = |S_2|$ is achieved for $S_1 \cup S_2 = \{1,\dots,n\}$, and it is $O(\sum_{i=1}^{n} [D(t,i,k-1)+x]^{\phi-2})$. ∎

LEMMA 24. *For all* $0 \le t \le T-1$, *and all* $0 \le k \le \log T$,

$$\sum_{i=1}^{n} [D(t,i,k)]^{\phi-2} \le (c \cdot \log T)^{\frac{\phi}{2}-1} \cdot n,$$

where c *is the constant of Lemma* 23.

Proof. To obtain a contradiction assume that there exists $0 \le t \le T-1$ and $0 < k \le \log T$ for which the bound does not hold. Consider the minimum such k. By Lemma 23 and the minimality of k, we get that $POT(t,k) \le POT(t,k-1) + c \cdot \sum_{i=1}^{n} D(t,i,k-1)^{\phi-2}$. Reapplying Lemma 23 and since the function $D(t,i,k)$ is increasing in k we get $POT(t,k) \le kc \cdot \sum_{i=1}^{n} D(t,i,k-1)^{\phi-2}$. Finally by the minimality of k, $POT(t,k) \le kc \cdot (c \cdot \log T)^{\frac{\phi}{2}-1} \cdot n = c^{\frac{\phi}{2}} \cdot (\log T)^{\frac{\phi}{2}-1} \cdot n \cdot k$. By our definition $POT(t,k) = \sum_{i=1}^{n} D(t,i,k)^{\phi}$. By Hölder inequality

$$\sum_{i=1}^{n} D(t,i,k)^{\phi} \ge n \left(\frac{\sum_{i=1}^{n} D(t,i,k)^{\phi-2}}{n} \right)^{\frac{\phi}{\phi-2}}.$$

However, by our assumption

$$n \left(\frac{\sum_{i=1}^{n} D(t,i,k)^{\phi-2}}{n} \right)^{\frac{\phi}{\phi-2}} > n \left(\frac{(c \cdot \log T)^{\frac{\phi}{2}-1} n}{n} \right)^{\frac{\phi}{\phi-2}} = (c \cdot \log T)^{\frac{\phi}{2}} \cdot n.$$

Combining the two inequalities we get

$$(\log T)^{\frac{\phi}{2}} < (\log T)^{\frac{\phi}{2}-1} \cdot k.$$

But this inequality implies that $k > \log T$, which is a contradiction. □

THEOREM 25. *The maximum drift is bounded by* $O(\sqrt{\log T} \cdot n^{\epsilon})$ *for any fixed* ϵ.

Proof. By Lemmas 23 and 24, the potential $POT(t, \log T)$ for all $1 \leq t \leq T$ is bounded by $\log T \cdot O((\log T)^{\frac{\phi}{2}-1} \cdot n) = O((\log T)^{\frac{\phi}{2}} \cdot n)$. This implies the bound on each drift, since the potential is the sum of the drifts to the power of ϕ. The constant ϵ is chosen to be $\frac{1}{\phi}$. □

5. Realizing frequencies rigidly. In this section we show how to construct a schedule that 12-approximates any feasible frequency vector in a *rigid* fashion on an interval graph. We reduce our rigid schedule problem to the dynamic storage allocation problem. The dynamic storage allocation problem is defined as follows. We are given objects to be stored in a computer memory. Each object has two parameters: (i) its size, that is, the number of cells needed to store it, and (ii) the time interval in which it should be stored. Each object must be stored in adjacent cells. The problem is to find the minimal size memory that can accommodate at any given time all of the objects that are needed to be stored at that time. The dynamic storage allocation problem is a special case of the multicoloring problem on interval graphs defined below.

A *multicoloring* of a weighted graph G with the weight function $w : V \to \mathcal{N}$ is a function $F : V \to 2^{\mathcal{N}}$ such that (i) for all $v \in V$ the size of $F(v)$ is $w(v)$, and (ii) if $(v, u) \in E$ then $F(v) \cap F(u) = \emptyset$. The multicoloring problem is to find a multicoloring with minimal number of colors. This problem is known to be an NP-Hard problem [GJ79].

Two interesting special cases of the multicoloring problem are when the colors of a vertex must be either contiguous or "spread well" among all colors. We call the first case the cont-MC problem and the second case the spread-MC problem. More formally, in a solution to cont-MC if $F(u) = \{x_1 < \cdots < x_k\}$, then $x_{i+1} = x_i + 1$ for all $1 \leq i < k$. Whereas in a solution to spread-MC that uses T colors, if $F(u) = \{x_1 < \cdots < x_k\}$ then (i) k divides T and (ii) $x_{i+1} = x_i + T/k$, for all $1 \leq i < k$, and $x_k + T/k - T = x_1$.

It is not hard to verify that for interval graphs the cont-MC problem is equivalent to the dynamic storage allocation problem described above. Simply associate each object with a vertex in the graph and give it a weight equal to the number of cells it requires. Put an edge between two vertices if their time intervals intersect. The colors assigned to a vertex are interpreted as the cells in which the object is stored.

On the other hand, the spread-MC problem corresponds to the rigid schedule problem as follows. First, we replace the frequency $f(v)$ by a weight $w(v)$. Let $k(v) = \lceil -\log_2 f(v) \rceil$, and let $k = \max_{v \in V}\{k(v)\}$; then $w(v) = 2^{k-k(v)}$. Clearly, $f(v)/2 \leq w(v)/2^k \leq f(v)$. Now, assume that the output for the spread-MC problem uses T colors, and let the colors of v be $\{x_1 < \cdots < x_k\}$ where $x_2 - x_1 = \Delta$. We interpret this as follows: v is scheduled in times $x_1 + i\Delta$ for all $i \geq 0$. It is not difficult to verify that the resulting schedule is rigid and it 2-approximates the given frequencies.

Although the dynamic storage allocation problem is a special case of the multicoloring problem it is still known to be an NP-Hard problem [GJ79]. Using similar arguments it can be shown that the rigid scheduling problem is also NP-Hard. Therefore, we are looking for an approximation algorithm. In what follows we present an

approximation algorithm that produces a rigid scheduling that 12-approximates the given frequencies. For this we consider instances of the cont-MC and spread-MC problems in which the input weights are powers of two.

DEFINITION 26. *A solution for an instance of* cont-MC *is both* aligned *and* contiguous *if for all* $v \in V$, $F(v) = \{j \cdot w(v), \ldots, (j+1) \cdot w(v) - 1\}$ *for some* $j \geq 0$.

In [Kie91], Kierstead presents an algorithm for cont-MC that has an approximation factor 3. A careful inspection of this algorithm shows that it produces solutions that are both aligned and contiguous for all instances in which the weights are powers of two.

We show how to translate a solution for such an instance of the cont-MC problem that is both aligned and contiguous into a solution for an instance of the spread-MC problem with the same input weights.

For $0 \leq x < 2^k$, let $\pi(x)$ be the k-bit number the binary representation of which is the inverse of the binary representation of x.

PROPOSITION 27. *For* $1 \leq i \leq k$ *and* $0 \leq j < 2^{k-i} = \Delta$, $\{\pi(j2^i), \ldots, \pi(j2^i + 2^i - 1)\} = \{\pi(j2^i), \pi(j2^i) + \Delta, \ldots, \pi(j2^i) + (2^i - 1)\Delta\}$.

This proposition says that an output of cont-MC that uses c colors can be transformed into an output of spread-MC that uses at most $2c$ colors.

Consider an instance of the spread-MC problem in which all the input weights are powers of two. Apply the solution of Kierstead [Kie91] to solve the cont-MC instance with the same input. This solution is both aligned and contiguous and uses at most $3T'$ colors where T' is the number of colors needed by an optimal coloring. Let $T \geq 3T'$ be the smallest power of 2 that is greater than T'. It follows that $T \leq 6T'$. Applying the transformation of Proposition 27 on the output of the solution to cont-MC yields a solution to spread-MC with at most T colors. This in turn yields an approximation factor of at most 12 for the rigid scheduling problem, since $w(v)/T \geq f(v)/2$.

THEOREM 28. *The above algorithm computes a rigid schedule that* 12*-approximates any feasible frequency vector on an interval graph.*

6. Circular-arc graphs. In this section we show how to transform any algorithm $\mathcal{A}$ for computing a schedule that c-approximates any given feasible frequency vector on interval graphs into an algorithm $\mathcal{A}'$ for computing a schedule that $2c$-approximates any given feasible frequencies on circular-arc graphs.

Let $\hat{f}$ be a feasible frequency vector on a circular-arc graph G.

Step 1. Find the maximum clique C in G.

Let $G' = G - C$. Note that G' is an interval graph. Let $\hat{g_1}$ and $\hat{g_2}$ be the frequency vectors resulting from restricting $\hat{f}$ to the vertices of G' and C, respectively. Note that $\hat{g_1}$ and $\hat{g_2}$ are feasible on G' and C, respectively.

Step 2. Using $\mathcal{A}$, find schedules S_1 and S_2 that c-approximate $\hat{g_1}$ and $\hat{g_2}$ on G' and C, respectively.

Step 3. Interleave S_1 and S_2.

Clearly, the resulting schedule $2c$-approximates $\hat{f}$ on the circular-arc graph G. Note also that all the three steps can be computed in polynomial time.

7. Conclusions and future research. In this paper we have introduced a new scheduling problem. It is characterized by the persistence and interdependency of the tasks involved. We have developed new measures that quantify the fairness and regularity of a schedule. We have shown that every conflict graph has a unique max-min fair frequency assignment and that, in general, this assignment is hard even to approximate. However, for perfect graphs, it turns out that max-min fair frequencies

are easy to compute and we have given an algorithm for this purpose. The scheduling algorithms described in this paper exhibit a tradeoff between the accuracy with which given frequencies are realized and their regularity. Furthermore, we have shown that a drift of one (i.e., P-fairness) is not achievable even for simple interval conflict graphs. This can be viewed as an indication that the problem in this paper is inherently more complex than the one considered in [BCPV96].

Many open problems remain. The exact complexity of computing a max-min fair frequency assignment in general graphs is not known, and there is no characterization of when such an assignment is easy to compute. All the scheduling algorithms in the paper use the inherent linearity of interval or circular-arc graphs. It would be interesting to find scheduling algorithms for the wider class of perfect graphs. The algorithm for interval graphs that realizes frequencies exactly exhibits a considerable gap in its drift. It is not clear from which direction this gap can be closed.

Our algorithms assume a central scheduler that makes all the decisions. Both from a theoretical and practical point of view it is important to design scheduling algorithms working in more realistic environments such as high-speed local-area networks and wireless networks (as mentioned in section 1.1). The distinguishing requirements in such an environment include a distributed implementation via a local signaling scheme, a conflict graph which may change with time, and restrictions on space per node and size of a signal. The performance measures and general setting, however, remain the same. A first step toward such algorithms has been recently carried out by Mayer, Ofek, and Yung in [MOY96].

Appendix A. The partial order $\prec$. In this appendix we prove that the relation $\prec$ is a partial order. We first observe that the definition can be restated as $\hat{f} \prec \hat{g}$ if there exists an index i and a threshold f such that $f_i < f \leq g_i$ (the *index* property), and for all $1 \leq j \leq n$, $g_j \geq \min\{f, f_j\}$ (the *threshold* property). The following two claims establish that $\prec$ is a partial order.

CLAIM 29. *The relation $\prec$ is antisymmetric.*

Proof. To obtain a contradiction assume that there exist two vectors $\hat{f}$ and $\hat{g}$ such that $\hat{f} \prec \hat{g}$ and $\hat{g} \prec \hat{f}$. This implies that there exist two indices i and ℓ and two thresholds f and g such that

1. $f_i < f \leq g_i$, and for all $1 \leq j \leq n$, $g_j \geq \min\{f, f_j\}$;
2. $g_\ell < g \leq f_\ell$, and for all $1 \leq j \leq n$, $f_j \geq \min\{g, g_j\}$.

Since $g_\ell \geq \min\{f, f_\ell\}$, $f_\ell > g_\ell$, and $g > g_\ell$, it follows that $g > f$. Similarly, since $f_i \geq \min\{g, g_i\}$, $g_i > f_i$, and $f > f_i$, it follows that $f > g$. We get the contradiction. □

CLAIM 30. *The relation $\prec$ is transitive.*

Proof. Suppose that $\hat{f} \prec \hat{g}$ and $\hat{g} \prec \hat{h}$. We show that $\hat{f} \prec \hat{h}$. Since $\hat{f} \prec \hat{g}$ and $\hat{g} \prec \hat{h}$ there exist two indices i and ℓ and two thresholds f and g such that

1. $f_i < f \leq g_i$, and for all $1 \leq j \leq n$, $g_j \geq \min\{f, f_j\}$;
2. $g_\ell < g \leq h_\ell$, and for all $1 \leq j \leq n$, $h_j \geq \min\{g, g_j\}$.

We choose $h = \min\{f, g\}$ as the threshold for $\hat{f} \prec \hat{h}$. Now, for all $1 \leq j \leq n$,

$$h_j \geq \min\{g, g_j\} \geq \min\{g, \min\{f, f_j\}\} \geq \min\{\min\{f, g\}, f_j\} \geq \min\{h, f_j\} .$$

We still have to prove that there exists an index with the desired property. Assume first that $h = f \leq g$; then we choose i as the index and we need to show that $f_i < h \leq h_i$. Since $h = f$ it follows that $f_i < h$. Since $h_i \geq \min\{g_i, g\}$, $h \leq g$, and $g_i \geq f = h$, it follows that $h_i \geq h$. Now assume that $h = g < f$; then we choose ℓ

as the index. Here we need to show that $f_\ell < h \le h_\ell$. Since $g \le h_\ell$ it follows that $h \le h_\ell$. Since $g_\ell \ge \min\{f, f_\ell\}$, $g > g_\ell$, and $h = g < f$, it follows that $h > f_\ell$. ∎

Appendix B. The complete proof of Theorem 5. We complete the proof of Theorem 5 for the case when $\hat{f}$ is not a rational convex combination.

CLAIM 31. *If a frequency vector $\hat{f}$ can be expressed as a convex combination of the independent sets, then $\hat{f}$ is feasible.*

Proof. Suppose that there exist weights $\{\alpha_I\}_{I\in\mathcal{I}}$ such that $\sum_{I\in\mathcal{I}} \alpha_I = 1$ and $\sum_{I\in\mathcal{I}} \alpha_I \chi(I) = \hat{f}$. We show how to obtain a schedule S that realizes the frequency vector $\hat{f}$. For every $k < \infty$, we pick $g_i^{(k)}$ to be a rational number between $f_i - 2^{-k}$ and f_i and apply Claim 7 to construct a schedule A_k of finite length, denoted $T(A_k)$, that realizes the frequency vector $\hat{g^{(k)}}$.

We go on to construct schedules $S_1, S_2, \ldots, S_k$ with the following properties.

Property 1. Schedule S_k has finite length $T(S_k)$.

Property 2. For each task $1 \le i \le n$, schedule S_k achieves a frequency of at least $f_i - 2^{-(k-1)}$ for task i.

Property 3. Schedule S_{k-1} is a prefix of schedule S_k.

Property 4. In the infinite schedule $S_kS_kS_k\cdots$ (i.e., the schedule given by concatenating the schedule S_k infinitely many times), for any task $1 \le i \le n$ and time $t > T(S_k)$, $f_i^{(t)}(S_kS_kS_k\cdots) \ge f_i - 2^{-(k-2)}$. (Recall that $f_i^{(t)}(S)$ is the prefix frequency of task i at time t in schedule S.)

Property 5. For any task $1 \le i \le n$ and time $t > T(S_{k-1})$, $f_i^{(t)}(S_k) \ge f_i - 2^{-(k-3)}$.

We construct the S_k's inductively. The base case S_1 exists trivially (every nonempty schedule satisfies the required properties). Assume the schedules $S_1, \ldots, S_{k-1}$ exist. Schedule S_k is given by the concatenation of n_1 schedulings of S_{k-1} followed by n_2 schedulings of A_k. We now show that under an appropriate choice of n_1 and n_2, the schedule S_k satisfies the above properties. Let D_i be the maximum among the drift of task i in the schedule S_{k-1} and the drift of task i in the schedule A_k. Let $D = \max_i\{D_i\}$. Let

$$n_1 = \left\lceil \frac{2^k D}{4T(S_{k-1})} \right\rceil \text{ and } n_2 = \left\lceil \frac{2n_1 T(S_{k-1})}{T(A_k)} \right\rceil.$$

Property 1. The period of S_k is

$$T(S_k) = n_1 T(S_{k-1}) + n_2 T(A_k)$$

which is finite since n_1 and n_2 are finite.

Property 2. The frequency of task i in S_k is at least

$$\frac{n_1 T(S_{k-1})(f_i - 2^{-(k-2)}) + n_2 T(A_k)(f_i - 2^{-k})}{n_1 T(S_{k-1}) + n_2 T(A_k)}$$

$$= f_i - 2^{-(k-1)} - \frac{n_1 T(S_{k-1}) 2^{-(k-1)} - n_2 T(A_k) 2^{-k}}{n_1 T(S_{k-1}) + n_2 T(A_k)}.$$

We wish to show that the above quantity is at least $f_i - 2^{-(k-1)}$. This simplifies to

$$n_2 \ge \frac{2n_1 T(S_{k-1})}{T(S_k)},$$

a condition which is satisfied by our choice of n_1 and n_2.

Property 3. Since $n_1 \geq 1$, it follows that S_{k-1} is a prefix of S_k.

Property 4. Since S_{k-1} is a prefix of S_k it follows that in the infinite schedule $S_kS_kS_k\ldots$, for any task $1 \leq i \leq n$ and time $t > T(S_k)$,

$$\begin{aligned} f_i^{(t)}(S_kS_kS_k\cdots) &= (f_i - 2^{-(k-1)})T(S_k) + (f_i - 2^{-(k-2)})[t - T(S_k)] - D_i \\ &= (f_i - 2^{-(k-2)})t + 2^{-(k-1)}T(S_k) - D_i. \end{aligned}$$

We wish to show that this is at least $t(f_i - 2^{-(k-2)})$. This condition simplifies to

$$2^{k-1}D_i \leq T(S_k) = n_1T(S_{k-1}) + n_2T(S_k).$$

Once again, the choice of n_1 and n_2 satisfies this condition.

Property 5. For any task $1 \leq i \leq n$ and time $T(S_{k-1}) < t \leq n_1T(S_{k-1})$, Property 4 of schedule S_{k-1} guarantees that $f_i^{(t)}(S_k) \geq f_i - 2^{-(k-3)}$. Now, consider $t > n_1T(S_{k-1})$. The number of times a task i is scheduled in S_k by time t is at least

$$(f - 2^{-(k-2)})n_1T(S_{k-1}) + (f - 2^{-k})(t - n_1T(S_{k-1})) - D_i$$

$$= (f - 2^{-(k-3)})t + 2^{-(k-2)}n_1T(S_{k-1}) + 7 \cdot 2^{-k}(t - n_1T(S_{k-1})) - D_i.$$

We wish to show that this quantity is at least $t(f-2^{-(k-3)})$. This inequality is implied by the condition $4n_1T(S_{k-1}) \geq 2^kD_i$, which is satisfied by the choice of n_1.

We use the sequences $S_1, \ldots, S_k, \ldots$ to define an infinite sequence S (which is essentially the limiting element of the sequence $\{S_i\}$). To determine which independent set to schedule at time t in S, we let k be the smallest index such that $T(S_k) \geq t$. We schedule the independent set scheduled by S_k at time t.

To see that S realizes the desired frequency vector $\hat{f}$, we prove that for every $\epsilon > 0$ there exists $T < \infty$ such that for all $t \geq T$ and for all tasks $1 \leq i \leq n$, $f_i^{(t)}(S) \geq f_i - \epsilon$.

Given $\epsilon > 0$, let k be the minimum integer such that $2^{-(k-2)} \leq \epsilon$ and let $T = T(S_k) + 1$. Given $t \geq T$, let k' be the largest index such that $t > T(S_{k'})$. Clearly, $k' \geq k$. Observe that for any $j < \infty$, S_j is a prefix of S. Thus, the prefix of schedule S up to time t is a prefix of $S_{k'+1}$. By Property 5 of $S_{k'+1}$, $f_i^{(t)}(S_{k'+1}) \geq f_i - 2^{-(k'-2)} \geq f_i - 2^{-(k-2)} \geq f_i - \epsilon$. □

Acknowledgment. We would like to thank Don Coppersmith and Moti Yung for many useful discussions.

REFERENCES

[AS90] B. Awerbuch and M. Saks, *A dining philosophers algorithm with polynomial response time*, in Proc. 31st IEEE Symp. on Foundations of Computer Science, 1990, pp. 65–75.

[BCPV96] S. Baruah, N. Cohen, C. Plaxton, and D. Varvel, *Proportionate progress: A notion of fairness in resource allocation*, Algorithmica, 15 (1996), pp. 600–625.

[BG87] D. Bertsekas and R. Gallager, *Data Networks*, Prentice–Hall, Englewood Cliffs, NJ, 1987.

[BP92] J. Bar-Ilan and D. Peleg, *Distributed resource allocation algorithms*, in Proc. 6th International Workshop on Distributed Algorithms, 1992, pp. 277–291.

[CCO93] J. Chen, I. Cidon, and Y. Ofek, *A local fairness algorithm for Gigabit LANs/MANs with spatial reuse*, IEEE J. on Selected Areas in Communication, 11 (1993), pp. 1183–1192.

[CM84] K. Chandy and J. Misra, *The drinking philosophers problem*, ACM Trans. on Programming Languages and Systems, 6 (1984), pp. 632–646.

[CO93] I. Cidon and Y. Ofek, *MetaRing – A full-duplex ring with fairness and spatial reuse*, IEEE Trans. on Communications, 41 (1993), pp. 110–120.

[CS92] M. Choy and A. Singh, *Efficient fault tolerant algorithms for resource allocation in distributed system*, in Proc. 24th ACM Symp. on Theory of Computing (1992), pp. 593–602.

[Dijk71] E. W. Dijkstra, *Hierarchical ordering of sequential processes*, Acta Informatica, 1 (1971), pp. 115–138.

[GJ79] M. Garey and D. Johnson, *Computers and Intractability, a Guide to the Theory of Np-completeness*, W. H. Freeman, San Francisco, 1979.

[Gol80] M. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[GLS87] M. Grötschel, L. Lóvasz, and A.Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, Berlin, 1987.

[Goo90] D. J. Goodman, *Cellular packet communications*, IEEE Trans. on Communications, 38 (1990), pp. 1272–1280.

[Jaf81] J. Jaffe, *Bottleneck flow control*, IEEE Trans. on Communications, 29 (1981), pp. 954–962.

[Kie91] H. A. Kierstead, *A polynomial time approximation algorithm for dynamic storage allocation*, Discrete Math., 88 (1991), pp. 231–237.

[Knu94] D. E. Knuth, *The sandwich theorem*, Electron. J. Combin., 1 (1994), pp. 1–48.

[LL73] C. L. Liu and J. W. Layland, *Scheduling algorithms for multiprogramming in a hard-real-time environment*, J. Assoc. Comput. Mach., 20 (1973), pp. 46–61.

[LY93] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, in Proc. 25th ACM Symp. on Theory of Computing, 1993, pp. 286–293.

[Lyn80] N. Lynch, *Fast allocation of nearby resources in a distributed system*, in Proc. 12th ACM Symp. on Theory of Computing, 1980, pp. 70–81.

[MOY96] A. Mayer, Y. Ofek, and M. Yung, *Local scheduling with partial state information for approximate max-min fair rates*, in Proc. IEEE INFOCOM'96, 1996.

[Tuc71] A. Tucker, *Matrix characterizations of circular-arc graphs*, Pacific J. Math., 39 (1971), pp. 535–545.

SIAM J. COMPUT.
Vol. 27, No. 4, pp. 1190–1202, August 1998
© 1998 Society for Industrial and Applied Mathematics
014

TIME–SPACE LOWER BOUNDS FOR DIRECTED *st*-CONNECTIVITY ON GRAPH AUTOMATA MODELS*

GREG BARNES† AND JEFF A. EDMONDS‡

Abstract. Directed *st*-connectivity is the problem of detecting whether there is a path from a distinguished vertex s to a distinguished vertex t in a directed graph. We prove time–space lower bounds of $ST = \Omega(\frac{n^2 \log n}{\log(n \log n/S)})$ and $S^{\frac{1}{2}}T = \Omega(m(n \log n)^{\frac{1}{2}})$ for directed *st*-connectivity on Cook and Rackoff's jumping automaton for graphs (JAG) model [*SIAM J. Comput.*, 9(1980), pp. 636–652], where n is the number of vertices and m the number of edges in the input graph, S is the space, and T the time used by the JAG. These lower bounds are simple and elegant, they approach the known upper bound of $T = O(m)$ when S approaches $\Theta(n \log n)$, and they are the first time–space tradeoffs for JAGs with an unrestricted number of jumping pebbles.

Key words. time–space tradeoffs, lower bounds, JAG graph *st*-connectivity

AMS subject classifications. 68Q05, 68Q15, 68Q25

PII. S0097539795294402

1. Introduction.

The *st*-connectivity problem is a fundamental one in computational complexity theory. The *st*-connectivity problem for *directed* graphs (STCON) is the prototypical complete problem for nondeterministic logarithmic space [20]. Both STCON and the corresponding problem for undirected graphs, USTCON, are DLOG-hard—any problem solvable deterministically in logarithmic space can be reduced to either problem [16, 20]. Understanding the complexity of *st*-connectivity is, therefore, a key to understanding the relationship between deterministic and nondeterministic space bounded complexity classes. For example, showing that there is no deterministic logarithmic space algorithm for directed connectivity would separate the classes DSPACE($\log n$) and NSPACE($\log n$), while devising such an algorithm would prove that DSPACE($f(n)$) = NSPACE($f(n)$) for any constructible $f(n) = \Omega(\log(n))$ [20]. Unfortunately, determining the complexity of STCON remains a difficult open problem. In this paper we devise *time–space tradeoffs* for STCON, that is, bounds on the *simultaneous* time and space requirements of algorithms for directed connectivity. Such tradeoffs are an important step toward solving the complexity of STCON. Time–space tradeoffs are also important in their own right, since they give more insight into the resource requirements of a problem or class of problems than a bound on time or space alone.

Proving lower bounds on the time or space requirements of STCON for a general model of computation, such as a Turing machine, is beyond the reach of current techniques. Thus, it is natural to consider a *structured* model [9] whose basic operations are based on the structure of the graph, as opposed to being based on the bits in the

*Received by the editors October 30, 1995; accepted for publication (in revised form) June 13, 1996; published electronically May 19, 1998. A preliminary version of this paper appeared as J. Edmonds's Ph.D. thesis, *Time–Space Lower Bounds for Undirected and Directed st-Connectivity on JAG Models*, University of Toronto, 1993.

http://www.siam.org/journals/sicomp/27-4/29440.html

†Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada (gsbarnes@plg.uwaterloo.ca). Portions of this work were performed while the author was at the Max-Planck-Institut für Informatik, Saarbrücken, Germany.

‡Department of Computer Science, York University, North York, Ontario, Canada M3J 1P3 (jeff@cs.yorku.ca, http://www.cs.yorku.ca/jeff). Portions of this work were performed while the author was at the University of Toronto, Canada.

graph's encoding. A natural structured model for the problem of *st*-connectivity is the *jumping automaton for graphs*, or *JAG*, introduced by Cook and Rackoff [11]. A JAG moves a set of pebbles on the graph. There are two basic operations—moving a pebble along a directed edge in the graph and jumping a pebble from its current location to the vertex occupied by another pebble. Although the JAG model is structured, it is not weak. In particular, it is general enough that most known deterministic algorithms for graph connectivity can be implemented on it. Poon [18] introduces the more powerful node-named JAG (*NNJAG*), an extension of the JAG model where the computation is allowed to depend on the names of the nodes on which the pebbles are located.

Cook and Rackoff [11] prove a lower bound of $\Omega(\log^2 n/\log\log n)$ on the space required for a JAG to compute directed *st*-connectivity (STCON). Berman and Simon [8] extend this result to randomized JAGs, and Poon [18] extends it to a probabilistic version of the NNJAG. Tompa [23] shows lower bounds on the product of the time and space needed when using certain natural approaches to solve STCON. Many time–space lower bounds have been proved for *undirected st*-connectivity (USTCON) on various weak versions of the JAG model [7, 10, 11]. Edmonds was the first to prove a time–space lower bound for USTCON on the unrestricted JAG model [13].

The standard algorithms for STCON, breadth- and depth-first search, run in optimal time $\Theta(m+n)$ and use $\Theta(n\log n)$ space. At the other extreme, Savitch's theorem [20] provides a small space ($\Theta(\log^2 n)$) algorithm that requires time exponential in its space bound (i.e., time $n^{\Theta(\log n)}$). Barnes et al. [3] show the first sublinear space, polynomial time algorithm for STCON. The algorithm runs in time $2^{O(\log^2(n/S))}\cdot n^3$ given space S. All of these algorithms can be implemented on the standard JAG [11, 19]. Using the NNJAG's ability to access the names of the nodes in the graph, Poon [18] shows how to implement Immerman's and Szelepcsényi's nondeterministic $O(\log n)$-space algorithm for directed *st*-nonconnectivity [15, 22] on a nondeterministic NNJAG. It is not clear that this algorithm can be implemented on a standard nondeterministic JAG.

The paper proves lower bounds of $ST=\Omega(\frac{n^2\log n}{\log(n\log n/S)})$ and $S^{\frac{1}{2}}T=\Omega(m(n\log n)^{\frac{1}{2}})$ for STCON on the JAG model, where $S\le 2n\log n$ is the space and T the time used by a JAG. The first bound is proved on directed graphs with outdegree at most three. Neither bound puts a restriction on the number of states used by the JAG. These lower bounds approach the known upper bound of $T=O(m)$ when S approaches $\Theta(n\log n)$ and are the first time–space tradeoffs for JAGs with an unrestricted number of jumping pebbles. An earlier version of this paper [4] proved a bound of $S^{\frac{1}{3}}T=\Omega(m^{\frac{2}{3}}n^{\frac{2}{3}})$ on the more powerful NNJAG model. This bound has since been improved to $T=2^{\Omega(\log^2(n/S))}$ for most S by Achlioptas, Edmonds, and Poon [1] and Edmonds and Poon [14], matching the previously mentioned upper bound of Barnes et al. [3]. Even though the bounds of the current paper are weaker than those in Edmonds and Poon, the results are still worth studying, because they apply to JAGs with an infinite number of states, because it is more likely they can be extended to the USTCON problem and because they are simple and elegant. In addition, this paper strengthens the model in two ways. Hence, technically speaking, the results here are not subsumed by the results of Edmonds and Poon.

In the following section, we formally define the JAG model. In section 3, we describe the families of *layered* and *comb graphs*, the graphs we use to prove our lower bounds. In section 4 we prove the $ST=\Omega(\frac{n^2\log n}{\log(n\log n/S)})$ and $S^{\frac{1}{2}}T=\Omega(m(n\log n)^{\frac{1}{2}})$

lower bounds for the JAG model. Finally, section 5 presents some notes and a discussion of future work.

For a survey of the graph connectivity problem, see Wigderson [24].

2. Definitions. A JAG [11] is a finite automaton with p distinguishable pebbles and q states. The input to a JAG is an instance of STCON $\langle G, s, t\rangle$, where G is a directed graph on n vertices with maximum outdegree d, and s and t are two distinguished nodes in the graph. For each node in the input graph, the outgoing edges are given a unique label in $\{1, \ldots, d\}$. The JAG begins its computation in state Q_0, with one of the pebbles on the distinguished node t and the other $p-1$ on s.

The program of the JAG may depend nonuniformly on n and on the degree d of the graph. What the JAG does each time step depends on the current state, the list of the pebbles that are on the distinguished vertices s and t, and the partition of the pebbles not on s and t, according to which pebbles are on the same vertices. Based on this information, the automaton changes state and either *walks* or *jumps* a pebble. Walking a pebble consists of selecting a pebble $P \in \{1, \ldots, p\}$ at some vertex v and some label $l \in \{1, \ldots, d\}$ and moving P along the edge out of v with label l. If there is no edge out of v with that label, the pebble stays at v. Jumping a pebble consists of selecting two pebbles $P, P' \in \{1, \ldots, p\}$ and moving P to the node occupied by P'. A JAG that solves STCON enters an accepting state if and only if there is a path from s to t in the input graph.

The space used by a JAG is defined to be $S = p \log_2 n + \log_2 q$, where p is the number of pebbles and q is the number of states. This corresponds to the $\log_2 n$ bits needed to store which of the n nodes a pebble is on and the $\log_2 q$ bits needed to record the current state.

This paper strengthens the definition of the JAG. First, it does not count the number of states as part of the space and hence applies even when the JAG has an arbitrarily large number of states. Second, it allows the pebbles to back up opposite the direction of the directed edge to the node that it came from. This is done by keeping on a stack the path of nodes taken from the source node s to its current position. We will refer to such a JAG as a *many states, stack JAG*.

This new JAG model is provably stronger than the original model. To begin, it is surprising that one can prove lower bounds when the number of states is allowed to be arbitrarily large. These states can be used to remember everything the JAG ever learns about the input graph. Previous study had indicated that as the number of states increases, the time to compute STCON on a JAG becomes linear. Evidently, this is not the case.

In addition, one complaint about JAGs is that they cannot traverse directed trees easily. Cook and Rackoff prove an $\Omega(\log^2 n/\log\log n)$ space for trees. However, a general model of computation is able to traverse trees easily in $O(\log n)$ space by means of depth first search. The JAG is unable to do this because it is unable to move a pebble to the parent of its current location as this would involve walking along the edge in the backward direction. Clearly, the many states, stack JAG is able to do this.

3. Comb and layered graphs. We prove the $ST = \Omega(\frac{n^2 \log n}{\log(n \log n/S)})$ lower bound on a class of graphs known as *layered graphs*. A layered graph consists of l layers (later set to $\log(\frac{n}{p \log(n/p)})$) of vertices, plus the extra distinguished vertex t. The number of vertices in layer i is χ_i and they are denoted $v_{\langle i,1\rangle}, v_{\langle i,2\rangle}, \ldots, v_{\langle i,\chi_i\rangle}$. The first layer is special: it contains $\chi_1 = cn$ vertices for some constant c; these vertices are

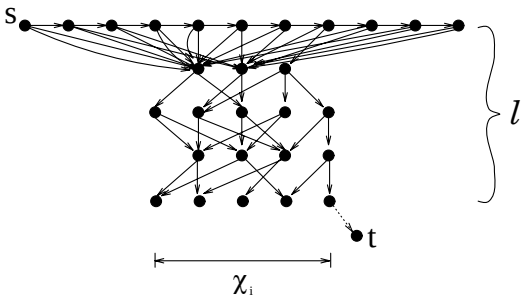


FIG. 1. *A layered graph.*

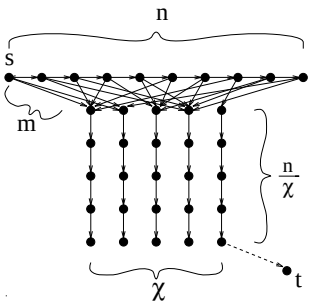


FIG. 2. *A comb graph.*

connected into a directed path using the *crossedges*, $(v_{\langle 1,1\rangle}, v_{\langle 1,2\rangle}), (v_{\langle 1,2\rangle}, v_{\langle 1,3\rangle}), \ldots, (v_{\langle 1,\chi-1\rangle}, v_{\langle 1,\chi_1\rangle})$; and the distinguished vertex s is the first vertex $v_{\langle 1,1\rangle}$ in this layer. Adjacent layers are connected as follows. Every vertex in layers 1 through $l-1$ has two *downedges* connecting it to two vertices on the next layer. We allow double edges, so it is of no concern if some vertex's two downedges go to the same vertex. Finally, there may or may not be an edge from a vertex on layer l to the distinguished vertex t. The edges are labeled in a straightforward way, say with 1 and 2 for the downedges of each node and 3 for the crossedges. See Figure 1 for an example of a layered graph. Note that no vertex in a layered graph has outdegree more than three, so the lower bound does not depend on the graph having a large number of edges.

We prove the $S^{\frac{1}{2}}T = \Omega(m(n\log n)^{\frac{1}{2}})$ lower bound on a different class of graphs known as *comb graphs*. A comb graph, illustrated in Figure 2, is composed of a *back*, χ *teeth*, and the distinguished node t. The back of the comb consists of a directed path of n nodes $v_1, \ldots, v_n$. The first node v_1 is the distinguished node s. The rth tooth consists of the directed path $u_{\langle r,1\rangle}, \ldots, u_{\langle r,l\rangle}$. The length of each tooth will be $l = \frac{n}{\chi}$ so that the total number of nodes in a comb graph is $2n+1$.

There are m $(\geq n)$ directed *connecting edges* $e_1, \ldots, e_m$ each going from a back node v_i to the top of one of the teeth in such a way that the outdegree of any two back nodes can differ by at most 1. The outdegree of the graph is then $\lceil \frac{m}{n} \rceil + 1$. More formally, for $j \in \{1, \ldots, m\}$, the connecting edge e_j is the $\lceil \frac{j}{n} \rceil$th edge emanating from back node $v_{1+(j-1) \bmod n}$ and has label $\lceil \frac{j}{n} \rceil$. We allow double edges, so it is of no concern if two edges from a back node go to the same tooth. If there is to be a directed path from s to t, then the node t is attached to the bottom of at least one of the teeth.

4. Lower bounds for JAGs. Intuitively, solving STCON for layered graphs is difficult because there are $cn \cdot 2^l$ possible paths from s to vertices on layer l. The JAG must potentially check each such path before it can be sure whether t is not connected to s. Of course, these paths will overlap in many places, but because the model is allocated a bounded amount of space, it is difficult for it to "remember" which subpaths have been traversed already. Therefore, many subpaths must be traversed many times before the JAG is sure they have all been traversed. Similarly, it is difficult for a JAG to "remember" which teeth in a comb graph have been traversed, so some teeth may get traversed many times before the JAG is sure that they all have been traversed.

There are two basic approaches to computing STCON on layered graphs. The first is the brute force approach. For each path from s to layer l, the JAG walks a pebble down the path. This requires at least $cn \cdot 2^l$ time steps.

The second approach is to learn enough about the downedges between the various layers, so that the subpaths starting at a vertex on layer i, for $i \in \{2, \ldots, l-1\}$, need to be traversed only once. For example, the JAG could move two pebbles to layer $i-1$ and then move the pebbles down two downedges, e and e', to layer i. If the pebbles collide, then e and e' point to the same vertex on layer i. With this knowledge, the JAG can avoid traversing all subpaths that begin with e', as long as it traverses all subpaths that begin with e. We show below that this strategy is not effective unless the JAG computes for at least $(\chi_i - 1)^2/(p-1)$ steps. In particular, if less time is used, there is a layered graph such that whenever the JAG moves two pebbles down two different edges to some layer, the pebbles never collide.

THEOREM 1. *Any many states, stack JAG that solves* STCON *on graphs with n vertices using p pebbles requires time $\Omega(\frac{n^2}{p \log(n/p)})$ (where the* log *on the bottom is at least* 1).

Since the space S used by a JAG is defined to be at least $p \log n$, the time–space tradeoff $ST = \Omega(\frac{n^2 \log n}{\log(n \log n/S)})$ follows. Note that the theorem sets no limit on the number of states in the JAG.

Proof. We will show that to solve STCON on layered graphs of size n, a JAG requires time $\mathrm{Min}(cn \cdot 2^l, c' \frac{n^2}{pl})$ for any $1 \le l \le c''n$ for some constants c, c', and c''. Putting $l = \log(\frac{n}{p \log(n/p)})$, we have $T = \Omega(\frac{n^2}{p \log(n/p)})$.

Suppose by way of contradiction that there is a JAG J that solves STCON on graphs of size n using p pebbles and using less time than the given bound. In order to bound how quickly the JAG can gain information, we will run J for this amount of time on graphs that have many more than n vertices. J is only supposed to run on graphs of size n, but because all the vertices in the input graph except s and t are indistinguishable during the computation of J, it is well defined how the computation would proceed if J were given a larger graph. We cannot expect J to solve STCON on this large graph, but we can run it for T time steps and see what happens.

More formally, the next move taken by a JAG is specified by the transition function. The input to this function is the following information: the current state, the list of the pebbles that are on the distinguished vertices s and t, and the partition of the pebbles not on s and t, according to which pebbles are on the same vertices. Call this information the current *configuration* of a JAG. A computation on an input graph is formally defined to be a sequence of such configurations. Given this definition, we can say that J's computation on two different graphs is identical, even if the graphs have a different number of vertices.

We run J on a set of larger graphs referred to as *k-tree graphs*, one k-tree graph for each $k \in \{1, \ldots, l\}$. A k-tree graph consists of a layered graph with k layers with the addition that each vertex $v_{\langle k,i \rangle}$ on the kth layer is the root of a directed binary tree of depth $l-k+1$, with edges directed down from the root. As with the layered graph, the downedges are labeled 1 and 2 and the crossedges are labeled 3. In addition, each k-tree graph has an isolated vertex t. The distinguished vertex s is defined to be $v_{\langle 1,1 \rangle}$ for every k-tree graph.

We prove by induction that for every $k \in \{1, \ldots, l\}$, there exists a k-tree G_k and a leaf vertex v_* of this graph such that during the computation of the JAG J on G_k, there is never a pebble on the vertex v_*. At the end of the proof, we need to find a graph with n vertices on which the JAG J gives the incorrect answer. We use the k-tree G_l, which is also a layered graph with n vertices, to find such a graph.

For the base case of the induction, $k = 1$, there is only one graph G_1 in the class of 1-trees, consisting of $\chi_1 = cn$ binary trees of depth l. This graph has $cn \cdot 2^l$ leaf vertices. Because JAG J uses fewer than $cn \cdot 2^l$ time steps, there must be some leaf vertex v_* that is never accessed in J's computation on G_1.

We are now ready to fix the number of vertices on each level of the k-tree graphs to be $\chi_i = \mathrm{Min}(2\chi_{i-1}, \sqrt{2(p-1)T_i}+1)$, where T_i is the number of time steps J walks a pebble from layer $i-1$ to layer i during this computation on the 1-tree. For each k, we will find a k tree on which the computation for J is identical as that on the 1-tree. Hence, T_i will be the number of time steps J walks a pebble from layer $i-1$ to layer i in each of these graphs.

For the inductive step, assume there is a $(k-1)$-tree, G_{k-1}, and a leaf v_* in G_{k-1} such that the computation of the JAG J on G_{k-1} never places a pebble on v_*. If χ_i is set to be $2\chi_{i-1}$, then we are done with the induction step, because G_{k-1} is already a k-tree with χ_i vertices at level i. Therefore, assume that $\chi_i = \sqrt{2(p-1)T_i}+1 < 2\chi_{i-1}$.

Think of G_{k-1} as follows. It has $k-1$ layers of a layered graph. Layer k has $2\chi_{i-1}$ vertices, the vertices in the second level of the binary trees rooted at layer $k-1$. Each of these vertices is the root of a directed binary tree of depth $l-k+1$. Denote these $2\chi_{i-1}$ disjoint binary trees by $\mathcal{T}_1, \ldots, \mathcal{T}_{2\chi_{i-1}}$. Denote the downedges going from layer $k-1$ to the roots of these trees by $e_1, \ldots, e_{2\chi_{i-1}}$.

The goal of the inductive step is to produce a k-tree G_k. Think of G_k as follows. Like G_{k-1}, it has $k-1$ layers of a layered graph. We will choose G_k so that G_{k-1} and G_k are identical on the first $k-1$ layers. Like G_{k-1}, G_k will have $2\chi_{i-1}$ downedges $e_1, \ldots, e_{2\chi_{i-1}}$ going from layer $k-1$ to layer k. Layer k of the k-tree, however, has only χ_i vertices, which are the roots of χ_i directed binary tree of depth $l-k+1$. Denote these χ_i binary trees by $\mathcal{T}'_1, \ldots, \mathcal{T}'_{\chi_i}$. What remains to be chosen in order to specify G_k are the connections between the downedges $e_1, \ldots, e_{2\chi_{i-1}}$, and the trees $\mathcal{T}'_1, \ldots, \mathcal{T}'_{\chi_i}$. These connections can be specified by choosing a partition of the trees $\mathcal{T}_1, \ldots, \mathcal{T}_{2\chi_{i-1}}$ into χ_i groups, $\mathcal{S}_1, \ldots, \mathcal{S}_{\chi_i} \subseteq \{\mathcal{T}_1, \ldots, \mathcal{T}_{2\chi_{i-1}}\}$. For $i \in \{1, \ldots, 2\chi_{i-1}\}$ and $h \in \{1, \ldots, \chi_i\}$, if $\mathcal{T}_i \in \mathcal{S}_h$, then the downedge e_i is connected to the root of the tree $\mathcal{T}'_h$ in the graph G_k. This can be thought of as collapsing the trees in the group $\mathcal{S}_h$ into the one tree $\mathcal{T}'_h$. See Figure 3.

We want to find a partition $\mathcal{S}_1, \ldots, \mathcal{S}_{\chi_i}$ with the property that the computation on the corresponding graph G_k is identical to that on G_{k-1}. Below, we show how to find a partition with the following property: for any two trees $\mathcal{T}_i$ and $\mathcal{T}_j$, if there is ever a time in the computation of J on G_{k-1} when one pebble is in $\mathcal{T}_i$ and another pebble is in $\mathcal{T}_j$, then these two trees will be in different groups in the partition. If this property is preserved, we can show that the sequence of configurations in the

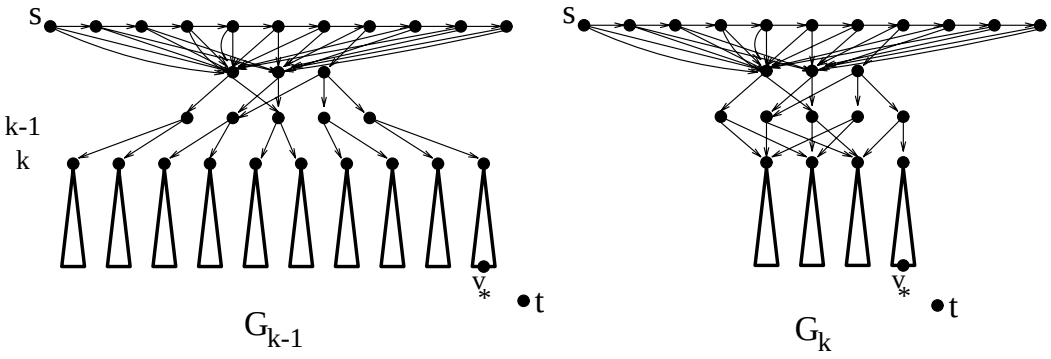


FIG. 3. *Collapsing the trees in G_{k-1} to form G_k.*

computation of J on G_{k-1} is the same as the sequence in the computation of J on a graph G_k. One difference between G_{k-1} and G_k is that in G_{k-1} the vertices in layer k all have indegree one. But the JAG model is defined so that it has no access to the indegree of a vertex. It is not hard to see that if the two computations were to deviate, the first deviation would occur because two pebbles collide in G_k that do not collide in G_{k-1}. To be more precise, if the two computations were to deviate, in the computation on G_k, one pebble must enter a tree T'_h via the downedge e_i, another pebble must enter the same tree via a different downedge e_j, and within this tree the two pebbles must meet. In the computation on G_{k-1}, which is the same up to this point, one pebble would enter the tree $\mathcal{T}_i$ via the downedge e_i, the other pebble would enter a different tree $\mathcal{T}_j$ via the downedge e_j, and clearly, these pebbles would not meet. Hence, the partition of the pebbles according to which pebbles are on the same vertices becomes different for the two computations. However, if we find a partition of the trees $\mathcal{S}_1, \ldots, \mathcal{S}_{\chi_i}$ with the desired property, such an event is not possible. It would mean that at some point during the computation on G_{k-1}, there is a pebble in the tree $\mathcal{T}_i$ and at the same time there is a pebble in the tree $\mathcal{T}_j$. By the property of the partition, these trees would be in different groups, e_i and e_j would be connected to different trees in G_k, and the pebbles entering these trees would not meet. It follows that the two sequences of configurations are the same.

The next step is to explain how a partition $\mathcal{S}_1, \ldots, \mathcal{S}_{\chi_i} \subseteq \{\mathcal{T}_1, \ldots, \mathcal{T}_{2\chi_{i-1}}\}$ with this property is found. Run the JAG J on the graph G_{k-1}, while maintaining an undirected graph H with vertex set $\{\mathcal{T}_1, \ldots, \mathcal{T}_{2\chi_{i-1}}\}$. The undirected edge $\{\mathcal{T}_i, \mathcal{T}_j\}$ is added to H if there is ever a time during the computation when one pebble is in the tree $\mathcal{T}_i$ and another pebble is in the tree $\mathcal{T}_j$.

We claim that H will contain at most $(p-1)T_i$ edges. A new edge can only be added to H if a pebble moves into some tree $\mathcal{T}_i$, i.e., a pebble is moved from layer $i-1$ to layer i. By definition, T_i is the number of such moves. During such a move, only one pebble is allowed to move. There are only $p-1$ other pebbles, so there are at most $p-1$ trees $\mathcal{T}_j$ already containing pebbles. Therefore, at most $p-1$ edges can be added to H at this step, one edge for each possible pair $\{\mathcal{T}_i, \mathcal{T}_j\}$. The following lemma shows that the chromatic number of H is then at most $\chi_i - 1$.

LEMMA 1. *Every undirected graph with no more than E edges has chromatic number at most $\sqrt{2E}$.*

H has at most $(p-1)T_i$ edges. Hence, by the lemma, it has chromatic number at most $\sqrt{2(p-1)T_i} = \chi_i - 1$.

Proof. Fix a graph. At most $\sqrt{2E}$ vertices have degree at least $\sqrt{2E}$. Give each of them its own color. The remaining vertices can be colored with the same $\sqrt{2E}$ colors—each vertex in turn is given a color that has not been assigned to one of its fewer than $\sqrt{2E}$ neighbors. □

Because H has chromatic number no more than $\chi_i - 1$, the vertices $\{\mathcal{T}_1, \ldots, \mathcal{T}_{2\chi_{i-1}}\}$ can be partitioned into $\chi_i - 1$ groups $\mathcal{S}_1, \ldots, \mathcal{S}_{\chi_i - 1}$ such that no edge of H has both ends in the same group. It follows that this partition has the required property.

To complete the induction step, we must find a leaf vertex of the k-tree that is never visited during the computation by J on the graph. Let $\mathcal{T}_*$ be the tree of G_{k-1} containing the leaf vertex v_*. Delete $\mathcal{T}_*$ from the group $\mathcal{S}_h$ that contains $\mathcal{T}_*$ and form a new group $\mathcal{S}_{\chi_i}$ containing only $\mathcal{T}_*$. This new partition also has the required property. Consider the leaf vertex of $\mathcal{T}'_*$ corresponding to the leaf vertex v_* of $\mathcal{T}_*$. The k-tree G_k defined by this new partition has the property that the only way to get from s to this leaf vertex is to traverse the downedge e_* and then to follow the path through the tree to the leaf. We can prove inductively that this path is unique and is defined by the same sequence of labeled edges in both G_{k-1} and G_k. Hence, it is reasonable to denote both the leaf vertex of G_{k-1} and this leaf vertex of G_k by v_*.

By the induction hypothesis, the computation of J on G_{k-1} never reaches the vertex v_*. By the stated property of the partition, the computation on G_k is identical to that on G_{k-1}. The same sequence of labels that must be traversed to reach v_* in G_k must be traversed to reach v_* in G_{k-1}. It follows that the computation on G_k never reaches the vertex v_*. This completes the inductive step.

After collapsing the k-tree graphs at each layer, we obtain a layered graph G_l. We claim that this graph contains at most n vertices. The number of vertices is $N = \sum_{i=1..l} \chi_i \le cn + \sum_{i=2..l} \sqrt{2(p-1)T_i} + 1$, where $\sum_{i=2..l} T_i = T \le c' \frac{n^2}{pl}$ is the total number of time steps. N is maximized when all the T_i's are equal to $\frac{T}{l-1}$. This gives $N \le cn + \sum_{k=2..l} \sqrt{2(p-1)c' \frac{n^2}{pl(l-1)}} + 1 \le cn + \sqrt{2c'}n + l - 1$, which is at most n for the appropriate choice of c, c', and c''.

By the induction proof, we know that there is a leaf vertex v_* in G_l that never contains a pebble during the computation of J. Let G'_l be the same graph as G_l except that there is a directed edge from the leaf v_* to the distinguished vertex t. Because J never places a pebble on vertex v_*, it can never detect whether there is an outgoing edge from v_* to t. Therefore, J's computation is the same on both G_l and G'_l, and hence J gives an incorrect answer for one of the graphs. Note that pebbles located on vertex t do not give the JAG any information about incoming edges. In fact, because t has no outgoing edges, pebbles on t can only move by jumping.

We now prove the second bound of $S^{\frac{1}{2}}T = \Omega(m(n \log n)^{\frac{1}{2}})$ for JAGs, using the comb graphs defined in section 3. Again, there are two basic approaches to computing STCON on comb graphs. One is the brute force approach. For each connecting edge, the JAG walks a pebble to the bottom of the tooth attached to it. This requires $m \times l$ time steps and two pebbles. The other approach is to learn enough about which connecting edges are attached to the same teeth, so that no tooth needs to be traversed more than once. The second approach requires $\Omega(\frac{\chi m}{p})$ time steps. This is proved by reducing the following *partition game* to the problem.

The partition game is parameterized by m, χ, and μ. The input consists of a partition of the edges $e_1, \ldots, e_m$ into χ nonempty groups. The player is able to specify two edges and query whether they are in the same group in the partition. The game is over when the player has determined a set C of μ or fewer edges that covers

the χ groups of the input partition; i.e., for each group, there is an edge in the group that is included in C.

LEMMA 2. *There are partitions for which the partition game requires at least* $\frac{1}{2}(\chi-1)(m-\mu)$ *queries.*

An upper bound of χm is easy for completely determining the partition. Query for each edge e_i whether e_1 and e_i are in the same group. These m queries determine all the edges that are in e_1's group in the partition. Delete these edges and repeat the process with the next edge, and so on.

Proof. The proof is by Impagliazzo [25]. The proof uses an adversary. The adversary maintains disjoint groups $P_1,\ldots,P_{\chi-1}\subseteq\{e_1,\ldots,e_m\}$ and an undirected graph H with m nodes, each node representing one of the connecting edges $e_i, 1\le i\le m$. (In the proof below, we refer to the connecting edges as nodes to avoid confusing them with the edges of H.) The adversary adds a node to the group P_r when it fixes the node to be in the rth group of the input partition, and it adds an edge $\{e_i,e_j\}$ to H when it reveals to the player that these nodes are in different groups. The adversary maintains the properties that two adjacent nodes in H are not in the same group and that the degree of every node in H that is not in one of the groups P_r is at most $\chi-2$.

When the player asks a question $\{e_i,e_j\}$ for the first time, the adversary does the following. For each of e_i and e_j, if it is not in some group P_r and has degree $\chi-2$ in H, then it is added to one of the groups that contains none of its neighbors in H. There are $\chi-1$ groups, so by the pigeonhole principle such a group exists. If e_i and e_j are both added to groups, it does not matter if they go into the same group. Now the adversary responds to the question. If e_i and e_j are in the same group, the adversary reveals this information. Otherwise, the adversary answers that e_i and e_j are in different groups and adds the edge $\{e_i,e_j\}$ to H.

A node is not added to a group until $\chi-1$ questions are asked about it. A single query involves two nodes, so $\frac{1}{2}(\chi-1)m'$ queries are required to place m' nodes in groups. Therefore, after $\frac{1}{2}(\chi-1)(m-\mu)-1$ queries, there are at least $\mu+1$ nodes not contained in any group P_r.

Assume the player completes the game with $\frac{1}{2}(\chi-1)(m-\mu)-1$ or fewer queries. At the end of the game, the player must specify a set C of μ nodes that covers each of the χ groups of the input partition. By the pigeonhole principle, there must be a node e_* that is among the $\mu+1$ nodes not contained in any group P_r and is not among the μ nodes in C specified by the player. The adversary then fixes the χth group P_χ to be a singleton group containing only e_*. Each of the nodes that has not yet been added to a group is added to one of the first $\chi-1$ groups that contains none of its neighbors in H (again, since these nodes have $\chi-2$ or fewer neighbors in H, such a group must exist). This defines a partition $P_1,\ldots,P_\chi$ that is consistent with all the answers given by the adversary. The player's set of nodes C must cover the χ groups, but it does not, since e_*, the only node in P_χ, is not in C. □

THEOREM 2. *Any many states, stack JAG that solves* STCON *on graphs with n vertices and m edges using p pebbles requires time $\Omega(mn^{\frac{1}{2}}/p^{\frac{1}{2}})$. We allow multiple edges. Hence, there is no restriction on m at all.*

As noted before, the space S used by a JAG is defined to be at least $p\log n$, so the time–space tradeoff $S^{\frac{1}{2}}T=\Omega(m(n\log n)^{\frac{1}{2}})$ follows.

Proof. We will show that to solve STCON on all comb graphs with n back nodes and m connecting edges requires time at least $\mathrm{Min}(\frac{m}{2}\times l, \frac{(\chi-1)(m/2)}{2(p-1)})$. If a JAG has

p pebbles, then setting χ to $n^{\frac{1}{2}}p^{\frac{1}{2}}$ gives the required bound $T = \Omega(mn^{\frac{1}{2}}/p^{\frac{1}{2}})$, since in a comb graph, $l = \frac{n}{\chi}$.

The proof reduces the above partition game to the STCON problem on comb graphs, where the parameter m in the game is the number of connecting edges in the comb graph, the parameter χ is the number of teeth in the comb graph, and the parameter μ is $\frac{m}{2}$. Suppose by way of contradiction that there is a JAG, J, that solves STCON on comb graphs with n back nodes and m connecting edges using p pebbles and in less than the stated time. Given J, we show that a player in the partition game can always beat the bound given in Lemma 2, a contradiction.

The game player beats the bound by simulating the execution of the JAG J on a comb graph G and using this simulation to construct its set C for the game. The graph G corresponds to the input partition in the game by partitioning the connecting edges into χ nonempty groups according to which edges point to which of the χ teeth. Note that the JAG cannot differentiate between the teeth. Hence, there is no particular order on the parts of the partition.

Initially, the game player knows neither the input partition nor the graph G. The player builds G "on the fly" in order to determine the information needed to simulate J's computation. This is done by repeatedly querying the input partition of the game and using the results of these queries to determine the structure of G and hence determine the next steps J will take. As J's computation proceeds, the player associates each pebble that is in a tooth with the connecting edge e_i through which it entered the tooth. If the pebble jumped into the tooth to pebble p_i, then it is associated with the connecting edge e_i that the pebble p_i was associated with when the jump occurred. Whenever there is one pebble associated with the connecting edge e_i and another pebble associated with e_j at the same time, J might learn whether e_i and e_j are connected to the same tooth. When this first happens, the game player queries $\{e_i, e_j\}$ and learns whether they are in the same group. If they are, the player implicitly connects the two connecting edges to the same tooth in the input graph, and if not, it implicitly connects them to two different teeth.

At any time t in the computation, G could be any graph in the set $\mathcal{G}_t$ of comb graphs that are consistent with the information revealed so far by the queries. The game player is able to continue this simulation on the incompletely specified G because J's computation is the same for every graph in $\mathcal{G}_t$. This follows from the following two observations. First, although the indegree of the nodes at the top of the teeth may be different for different graphs in $\mathcal{G}_t$, J has no access to the indegree of nodes. Second, when two pebbles enter teeth via two different connecting edges e_i and e_j, the answers to the queries ensure that they are either in the same tooth for every graph in $\mathcal{G}_t$ or in different teeth for every graph. Thus two pebbles meet during the computation on one of the graphs in $\mathcal{G}_t$ if and only if they meet during the computation on all such graphs.

The goal of the player is to construct the set C that covers the χ groups of the game's partition. Whenever there is a pebble associated with the connecting edge e_i that traverses down to the bottom of the tooth connected to e_i, the game player adds e_i to the set C. Intuitively, J must traverse all the teeth to be sure the input graph is not s-t connected, so the constructed set C must cover all the groups in the input partition to the game.

We now prove that given the JAG J, the player will never make more than $\frac{1}{2}(\chi - 1)(\frac{m}{2}) - 1$ queries during the computation and will add at most $\frac{m}{2}$ edges to C and that these edges will cover all groups in the game's input partition, which

will lead to a contradiction. First, at each step of J's computation only one pebble is allowed to move. This step causes the game player to make a query only if this pebble moves into the tooth attached to some edge e_i while there is another pebble already in a tooth attached to some other edge e_j. There are only $p-1$ other pebbles, so a step causes the game player to make at most $p-1$ queries. Since J uses no more than $\frac{(\chi-1)(\frac{m}{2})}{2(p-1)} - 1$ steps, no more than $\frac{1}{2}(\chi-1)(\frac{m}{2}) - 1$ queries will be made.

Second, the game player will add at most $\frac{m}{2}$ connecting edges to the set C. The number of JAG computation steps required to move a pebble into a tooth via a connecting edge and then to the bottom of the tooth is at least l, the length of the tooth. J's computation proceeds for fewer than $(\frac{m}{2})l$ steps, so this is done for at most $\frac{m}{2}$ connecting edges.

Finally, for every game input partition, the set C constructed by the player will cover all of the groups in the partition. By way of contradiction, suppose that for some partition there is a group P_r in the partition that is disjoint from C, and let the rth tooth be the one whose edges correspond to the group P_r. During the computation of J on G, a pebble can never traverse to the bottom of the rth tooth. If a pebble did, it must have entered the tooth via some connecting edge, and that edge must be in both C and P_r. It follows that a pebble is never on the bottom node $u_{\langle r,l\rangle}$ of this tooth.

Let G' be the same graph as G except that the node t is attached to the bottom of the rth tooth. J's computation is identical on the graphs G and G', because J must have a pebble on node $u_{\langle r,l\rangle}$ in order to know whether there is an outgoing edge from it to t (as noted in the proof of Theorem 1, pebbles located on node t do not give J any information about incoming edges). Because the computation is the same on G and G', the JAG J must give an incorrect answer for one of the graphs. But we assumed that J correctly solves STCON for all these graphs. Therefore, the set C will cover all the groups in the game's input partition. But this contradicts Lemma 2, since then the game player always finishes the game with fewer than the required number of queries. □

5. Open problems. The obvious open problems presented by this work are to improve the STCON lower bounds for the JAG and NNJAG. Subsequent to this work, Achlioptas, Edmonds, and Poon [1] and Edmonds and Poon [14] have shown a lower bound of time $T = 2^{\Omega(\log^2(n/S))}$ given space $S \leq n^{1-\Omega(1)}$ for the probabilistic NNJAG (or $T = 2^{\Omega(\log^2(n/S)/\log\log n)} \times (nS/\log n)^{\frac{1}{2}}$ otherwise). Not only does this greatly improve the current bounds, it shows that the upper bound of Barnes et al. [3] of time $2^{O(\log^2(n/S))} \times n^{O(1)}$ is optimal NNJAG.

There are a few directions in which these results could be extended. First, the small gap of $\log\log n$ between Barnes et al.'s upper bound and Edmonds and Poon's lower bound at the lower end of the space spectrum could be eliminated. Second, the lower or upper bounds for STCON could be improved at the higher end of the space spectrum.

Finally, better lower bounds are needed for the undirected version of s-t connectivity. There is a probabilistic upper bound of $S \cdot T \in m^{1.5}n^{.5}\log^{O(1)} n$ [5] that can be run on a JAG. It is interesting how close our lower bounds are to this upper bound. The current JAG lower bounds for USTCON are weak and only allow a small number of pebbles [13]. Our hope is that the techniques used in this paper can be applied to the undirected version of s-t connectivity. [12] discusses some ideas and difficulties in doing this.

Ultimately, one would like to prove lower bounds for STCON on a general model of computation. Any nontrivial bounds for general models would be a step in this direction. A more modest goal would be to add features to the JAG or NNJAG to make it more general (as Poon added node names to the JAG to devise the NNJAG [18]) and to prove the same bounds on these more general models.

Acknowledgments. We would like to thank Faith Fich for her extensive support and Paul Beame, Al Borodin, Russell Impagliazzo, and Hisao Tamaki for their helpful comments and suggestions. We would also like to thank a referee for adding another factor of $\log n$ to the first lower bound by having a different number of nodes χ_i on each level.

REFERENCES

[1] D. Achlioptas, J. Edmonds, and C. K. Poon, *Tight lower bounds for st-connectivity on NNJAGs*, SIAM J. Comput., to appear.

[2] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovász, and C. W. Rackoff, *Random walks, universal traversal sequences, and the complexity of maze problems*, in 20th Annual Symposium on Foundations of Computer Science, IEEE, San Juan, Puerto Rico, Oct. 1979, pp. 218–223.

[3] G. Barnes, J. F. Buss, W. L. Ruzzo, and B. Schieber, *A sublinear space, polynomial time algorithm for directed s-t connectivity*, in Proc., Structure in Complexity Theory, Seventh Annual Conference, IEEE, Boston, MA, June 1992, pp. 27–33; SIAM J. Comput., to appear.

[4] G. Barnes and J. A. Edmonds, *Time-space lower bounds for directed s-t connectivity on JAG models*, in Proc. 34th Annual Symposium on Foundations of Computer Science, IEEE, Palo Alto, CA, Nov. 1993, pp. 228–237.

[5] G. Barnes and U. Feige, *Short random walks on graphs*, in Proc. 25th Annual ACM Symposium on Theory of Computing, San Diego, CA, May 1993, pp. 728–737.

[6] G. Barnes and W. L. Ruzzo, *Deterministic algorithms for undirected s-t connectivity using polynomial time and sublinear space*, in Proc. 23rd Annual ACM Symposium on Theory of Computing, New Orleans, LA, May 1991, pp. 43–53; Department of Computer Science and Engineering, University of Washington Tech. report 91-06-02; Comput. Complexity 6 (1996–97), pp. 1–28.

[7] P. W. Beame, A. Borodin, P. Raghavan, W. L. Ruzzo, and M. Tompa, *Time-space tradeoffs for undirected graph traversal by graph automata*, Inform. Comput., 130 (1996), pp. 101–129.

[8] P. Berman and J. Simon, *Lower bounds on graph threading by probabilistic machines*, in Proc. 24th Annual Symposium on Foundations of Computer Science, IEEE, Tucson, AZ, Nov. 1983, pp. 304–311.

[9] A. Borodin, *Structured* vs. *general models in computational complexity*, L'Enseignement Mathématique, 28 (1982).

[10] A. Borodin, W. L. Ruzzo, and M. Tompa, *Lower bounds on the length of universal traversal sequences*, J. Comput. System Sci., 45 (1992), pp. 180–203.

[11] S. A. Cook and C. W. Rackoff, *Space lower bounds for maze threadability on restricted machines*, SIAM J. Comput., 9 (1980), pp. 636–652.

[12] J. A. Edmonds, *Time-Space Lower Bounds for Undirected and Directed ST-Connectivity on JAG Models*, Ph.D. thesis, Department of Computer Science, University of Toronto, Aug. 1993.

[13] J. A. Edmonds, *Time-space trade-offs for undirected st-connectivity on a JAG*, in Proc. 25th Annual ACM Symposium on Theory of Computing, San Diego, CA, May 1993, pp. 718–727; SIAM J. Comput., to appear.

[14] J. Edmonds and C. K. Poon, *A nearly optimal time-space lower bound for graph connectivity problem on NNJAG model*, in Proc. 27th Annual ACM Symposium on Theory of Computing, 1995, pp. 147–156.

[15] N. Immerman, *Nondeterministic space is closed under complementation*, SIAM J. Comput., 17 (1988), pp. 935–938.

[16] H. R. Lewis and C. H. Papadimitriou, *Symmetric space-bounded computation*, Theoret. Comput. Sci., 19 (1982), pp. 161–187.

[17] N. Nisan, E. Szemerédi, and A. Wigderson, *Undirected connectivity in $O(\log^{1.5} n)$ space*, in Proc. 33rd Annual Symposium on Foundations of Computer Science, IEEE, Pittsburgh, PA, Oct. 1992, pp. 24–29.

[18] C. K. Poon, *Space bounds for graph connectivity problems on node-named JAGs and node-oriented JAGs*, in Proc. 34th Annual Symposium on Foundations of Computer Science, IEEE, Palo Alto, CA, Nov. 1993.

[19] C. K. Poon, *A Sublinear Space, Polynomial Time Algorithm for Directed ST-Connectivity on the JAG Model*, Ph.D. thesis, University of Toronto, 1995.

[20] W. J. Savitch, *Relationships between nondeterministic and deterministic tape complexities*, J. Comput. System Sci., 4 (1970), pp. 177–192.

[21] W. J. Savitch, *Maze recognizing automata and nondeterministic tape complexity*, J. Comput. System Sci., 7 (1973), pp. 389–403.

[22] R. Szelepcsényi, *The method of forcing for nondeterministic automata*, Acta Inform., 26 (1988), pp. 279–284.

[23] M. Tompa, *Two familiar transitive closure algorithms which admit no polynomial time, sublinear space implementations*, SIAM J. Comput., 11 (1982), pp. 130–137.

[24] A. Wigderson, *The complexity of graph connectivity*, in Mathematical Foundations of Computer Science 1992: Proceedings, 17th Symposium, I. M. Havel and V. Koubek, eds., Lecture Notes in Computer Science 629, Springer-Verlag, Prague, Czechoslovakia, Aug. 1992, pp. 112–132.

[25] R. Impagliazzo, personal communication, 1993.

SIAM J. COMPUT.
Vol. 27, No. 4, pp. 1203–1220, August 1998
© 1998 Society for Industrial and Applied Mathematics
015

A CHERNOFF BOUND FOR RANDOM WALKS ON EXPANDER GRAPHS*

DAVID GILLMAN†

Abstract. We consider a finite random walk on a weighted graph G; we show that the fraction of time spent in a set of vertices A converges to the stationary probability $\pi(A)$ with error probability exponentially small in the length of the random walk and the square of the size of the deviation from $\pi(A)$. The exponential bound is in terms of the expansion of G and improves previous results of [D. Aldous, *Probab. Engrg. Inform. Sci.*, 1 (1987), pp. 33–46], [L. Lovász and M. Simonovits, *Random Structures Algorithms*, 4 (1993), pp. 359–412], [M. Ajtai, J. Komlós, and E. Szemerédi, *Deterministic simulation of logspace*, in Proc. 19th ACM Symp. on Theory of Computing, 1987].

We show that taking the sample average from one trajectory gives a more efficient estimate of $\pi(A)$ than the standard method of generating independent sample points from several trajectories. Using this more efficient sampling method, we improve the algorithms of Jerrum and Sinclair for approximating the number of perfect matchings in a dense graph and for approximating the partition function of a ferromagnetic Ising system, and we give an efficient algorithm to estimate the entropy of a random walk on an unweighted graph.

Key words. random walk, graph, eigenvalue, expander, large deviations, approximate counting, matching, Ising system, partition function, Markov source, entropy

AMS subject classifications. 60F10, 60J10, 62M05, 68Q25, 94A29

PII. S0097539794268765

1. Introduction.

Let G be a connected undirected graph with positive weights on the edges. We consider the random walk on G, which at each time step chooses an edge leaving the current vertex with probability proportional to the weight on the edge. The random walk converges to a limiting distribution π on the vertices of G.[1] This model is equivalent to a finite reversible Markov chain.

Let A be a subset of vertices of G. We consider the amount of time t_n the random walk spends in A during the first n steps. It is well known that for almost every trajectory of the random walk, the fraction of time spent in A, t_n/n, converges to the limiting probability of A, $\pi(A)$ [Do, Theorem 6.1].

We quantify this rate of convergence. We are concerned with the probability that for a given n-step trajectory of the random walk, t_n/n will deviate by some specified amount from $\pi(A)$ (the *deviation probability*). Theorem 2.1 shows that this probability decays exponentially in the square of the amount of deviation, as a multiple of $1/\sqrt{n}$. This bound is of a similar form to that given by Chernoff [Ch] for the case of independent random variables (a very special case of random walk).

The exponent in the bound of Theorem 2.1 is proportional to the eigenvalue gap of the transition matrix of the random walk, which is directly related to the expansion of G [AM, Ta, Alo, SJ]. The bound also depends on the starting distribution of the random walk, which may introduce a factor of $|G|$. In this case $O(\log|G|)$ random

*Received by the editors June 1, 1994; accepted for publication (in revised form) June 17, 1996; published electronically May 19, 1998. The research and writing of this paper were supported by NSF grant 9212184-CCR and DARPA contract N00014-92-J-1799.

http://www.siam.org/journals/sicomp/27-4/26876.html

†Iterated Systems, Inc., 3525 Piedmont Road, Building 7, Suite 600, Atlanta, GA 30305 (dgillman@iterated.com).

[1] We are interested in convergence in the following weak sense: let $P^{(k)}$ be the distribution of the random walk after k steps. Then $\frac{1}{n}\sum_{k=1}^{n} P^{(k)} \to \pi$. This holds even when G is bipartite.

walk steps suffice to get a good estimate of $\pi(A)$. When the random walk starts close to the stationary distribution the bound does not depend on $|G|$.

Theorem 2.1 is the first exponential bound on the deviation probability in terms of a computable quantity (in this case the eigenvalue gap). This result implies bounds on all higher moments of the fraction of time spent in A, and it quantifies the rate of convergence to $\pi(A)$ in each L^p norm, $1 \leq p < \infty$ [Kah]. It also sharpens a theorem of Ajtai, Komlós, and Szemerédi [AKS] (see also [CW] and [IZ]), which showed that the probability of a deviation of constant size decays exponentially in n. Aldous [Ald87] bounded the variance of the fraction of time spent in A in terms of the eigenvalue gap. Lovász and Simonovits [LS] gave a similar result for arbitrary measure spaces. Those results give quadratic bounds on the deviation probability via Chebyshev's inequality. Goldreich et al. [G*] have given a bound on the probability of not hitting A in n steps which decays exponentially in $n\pi(A)$.

Theorem 2.1 applies more generally to estimating the expectation of a nonnegative function on the vertices of G. $\pi(A)$ is the expectation of χ_A, the indicator function of A, and Theorem 2.1 states that the fraction of time spent in A is a good estimate of this expectation. We establish the analogous result for estimating the expectation $\mathrm{E}f$ of an arbitrary nonnegative function f on the vertices of G. In this case the bound depends on $\max_x |f(x)|$ as well as on the eigenvalue gap.

Approximation algorithms. Estimating $\pi(A)$ (or in some applications, $\mathrm{E}f$) is a fundamental problem for approximation algorithms in which A and G are exponentially large combinatorial sets such as sets of matchings of a graph [JS89]. The basic strategy is to generate random sample points in G and compute the fraction that are in A. The standard procedure is to use the rapid mixing property of the random walk on G to generate a single nearly random sample point from π. The random walk is repeated to generate the number of independent sample points Chernoff's bound requires [DFK, JS89, LS]. We compare this with the alternative procedure analyzed by Aldous in [Ald87], which was the first to generate a nearly random point in G and then to continue the random walk from that point, sampling every subsequent vertex. This procedure is commonly used in statistical biology and physics, usually without rigorous analysis of its reliability; for convenience we will refer to it as "Aldous's procedure" (AP).

We show in this paper that AP (sometimes in a modified form) requires fewer random walk steps than the standard procedure to ensure the same confidence in the resulting estimate of $\pi(A)$. A little intuition here will show why one would expect this to be the case. Let τ be the "relaxation time" of the random walk, which is the number of steps required to generate a single random point. The standard procedure picks up l sample points by taking τl random walk steps. We may as well assume that these sample points come from a single long random walk on the graph which visits the vertices $x_1, x_2, \ldots$. Then the sample points are $x_\tau, x_{2\tau}, \ldots, x_{l\tau}$. Now consider using AP on the same random walk. The estimate it gives is an average from the (highly dependent) sample points $x_{\tau+1}, x_{\tau+2}, \ldots, x_{l\tau}$. But this estimate is just an average of τ different estimates, each using the standard procedure with $l-1$ samples: for each fixed $i \leq \tau$ the ith estimate uses the sample points $x_{\tau+i}, x_{2\tau+i}, \ldots, x_{(l-1)\tau+i}$. It should not hurt each estimate much that it only samples $l-1$ points instead of l, whereas it should help that τ different (admittedly dependent) estimates are being averaged.

Stating the results more quantitatively, we analyze these procedures within the framework of (β, δ)-approximation algorithms; i.e., algorithms with input parameters

β and δ that with probability $1-\delta$ output an approximation of $\pi(A)$ with relative error β [KL]. We assume throughout that we can efficiently find one point $s \in G$ from which to start a random walk.

The main consequence of Theorem 2.1 for (β, δ)-approximation algorithms is that AP requires $O(\log(1/\pi(s)) + \log(1/\delta)/\beta^2)$ random walk steps. The first term is the relaxation time. The standard procedure (SP) requires $O(\log(1/\pi(s)) \log(1/\delta)/\beta^2)$. The constants in both cases depend on the eigenvalue gap and $\pi(A)$.

Unfortunately, the dependence on $\pi(A)$ favors the standard procedure when $\pi(A)$ is small (often in practice $\pi(A)$ is $O(1/n)$). This is because the bound in Theorem 2.1 is not optimal for small $\pi(A)$ in comparison with either the Chernoff bound for independent random variables or the variance bound of Aldous for Markov chains. For this reason the best procedure when $\pi(A)$ is small is to use AP with δ replaced by a constant, say $1/4$, to repeat this estimate $\log(1/\delta)$ times, and to take the median of the answers. The technique of using the median of several estimates is well known and was introduced by Jerrum, Valiant, and Vazirani in [JVV]. Our analysis of this modified procedure depends on an extension of the variance bound of Aldous.

We present a modified version of an approximation algorithm, due to Jerrum and Sinclair, for evaluating the partition function of a ferromagnetic Ising system. Our new version of the algorithm improves the running time from $O(|E|^3 m^7)$ to $O(|E|^2 m^6)$, where m is the number of vertices and E is the edge set of the system [JS91]. This algorithm makes calls to a (β, δ)-approximation routine for $\mathrm{E}f$ for different functions f. Part of the improved running time of the algorithm comes from replacing the standard sampling procedure by AP. The rest of the improvement comes from the observation that the errors from the different calls to the (β, δ)-approximation routine are independent and cancel one another out to some extent. Dyer and Frieze used the same observation in their algorithm for computing the volume of a convex body in Euclidean n-space [DF].

We are also able to improve an approximation algorithm, due to Jerrum and Sinclair, for counting the number of perfect matchings in a graph. Suppose the graph $H = (V, E)$ has $2m$ vertices. Our new version of the algorithm improves the running time from $O(q^3 m^6 |E| \log^2 m)$ to $O(q^2 m^5 |E| \log m)$, where q is an a priori upper bound on the ratio of the number of matchings with $m-1$ edges to the number of perfect (m-edge) matchings [JS89]. Polynomial bounds on q are known for large classes of graphs; for example, $q(m) = m^2$ for dense bipartite graphs and for regular periodic lattices [JS89, KRS]. Part of the improved running time of our new version of the algorithm comes from implementing AP. The algorithm makes calls to a $(\beta, O(1))$-approximation routine for different sets A. In addition, the new algorithm introduces a strategy of selecting only those sets A for which $\pi(A)$ is not too small. The importance of this is that the running time of the $(\beta, O(1))$-approximation routine depends inversely on $\pi(A)$.

Entropy estimation. In the special case where the edges of G are not weighted, Theorem 2.1 shows that one can compute the entropy of the random walk accurately from a very short realization of it. We view the random walk on G as an information source whose alphabet is the vertex set of G. It is convenient to state the result in this form, although it applies to any labelling of the vertices such that the Markov chain is unifilar. (A *unifilar* Markov chain is one in which each state has nonzero transition probability to at most one state of each label.) If G has constant eigenvalue gap and bounded degree then a good entropy estimate requires only $O(\log |G|)$ steps of the random walk.

This result quantifies the rate of convergence of the classical Shannon–McMillan asymptotic equipartition property [Ash]. The classical result assumes an ergodic information source with entropy H. It says, intuitively, that for large n, roughly 2^{Hn} of the n-bit strings each have probability roughly 2^{-Hn}. For purposes of encoding n-bit blocks of output from the source into blocks of some fixed shorter length greater than Hn, we give a bound on the exponent of the error probability. This is the first bound on the error exponent for fixed-length noiseless source coding in terms of a computable property of the underlying Markov chain. Previous bounds on the error exponent have been based on divergence and retain an asymptotic flavor. Large deviation methods for the source coding problem were used in [Nat, An].

Methods. The proof of Theorem 2.1 begins with Theorem 2.2, due to Höglund, which follows the method of Cramér [Cr] and Chernoff [Ch] of estimating the deviation probability in terms of the moment generating function of the number of visits to A [H, Theorem 5.5] (see also Nagaev [Nag, Theorem 6]). Höglund writes the moment generating function as an expression involving the largest eigenvalue of a perturbation of the transition matrix for the random walk. How hard it is to find a quantitative bound for this expression can depend on the transition matrix. Generally, the eigenvalues of a matrix may vary wildly under small perturbations of the matrix [SS, p. 166]. Höglund was able to use his method to derive a bound similar to Chernoff's for the case of Bernoulli trials (in which the transition matrix has identical rows).

Theorem 2.1 demonstrates the applicability of Höglund's approach to random walks on weighted graphs, where the transition matrix is similar to a symmetric matrix. We first bound the logarithm of the largest eigenvalue of a perturbation of the transition matrix by estimating its second derivative with respect to a perturbation parameter r. This bound uses Cauchy's estimate and the observation that the largest eigenvalue is an analytic function of r [Ahl, Kat]. The first bound leads to a bound on the probability of deviation in terms of r, and Theorem 2.1 follows by optimizing over r.

Overview. In section 2 we state and prove our main theorem. At the end of the section we compare similar recent results due to Kahale [Kah] and Dinwoodie [Di95a, Di95b]. In section 3 we give a general comparison of sampling procedures. There we prove an extension of the variance bound of Aldous (Proposition 3.2). In section 4 we describe an improved version of an approximation algorithm for the partition function of an Ising system, and we summarize our improvements to an algorithm for counting perfect matchings in a graph. Finally, in section 5 we discuss the use of random walks to estimate entropy of an information source generated by a random walk.

2. The main theorem. Let $G = (V, E)$ be a connected undirected graph. Let each edge $\{x, y\}$ in the edge set E be assigned a positive weight w_{xy}. We define the weight w_x of the vertex x by the formula $w_x = \sum_{\{x,y\}\in E} w_{xy}$.

A random walk on such a weighted graph is equivalent to a time-reversible finite Markov chain. The states of the Markov chain are the vertices of the graph. The Markov chain is defined by its *transition matrix* $P = (p_{xy})$; p_{xy} is the probability (independent of time) of moving to state y after entering state x, given by

$$p_{xy} = \begin{cases} \frac{w_{xy}}{w_x} & \text{if } \{x, y\} \in E, \\ 0 & \text{if not.} \end{cases}$$

By the classical theory of nonnegative matrices, the eigenvalues of P are $1 = \lambda_1 >$

$\lambda_2 \geq \cdots \geq \lambda_{|V|} \geq -1$, and $\lambda_{|V|} = -1$ if and only if G is bipartite [Se]. The strict separation of λ_1 and λ_2 follows from the connectedness of G. Let π denote the unique left eigenvector with eigenvalue 1. Properly normalized, $\pi(x) = w_x / \sum_{y \in V} w_y$ for all $x \in V$, and π is a probability distribution. We refer to π as the *stationary distribution*. For all probability distributions $\mathbf{d}$ on V, $\lim_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n-1} P^k \mathbf{d} = \pi$. (When G is not bipartite, we have the stronger property that $P^n \mathbf{d} \to \pi$ for all $\mathbf{d}$.) Let $\epsilon := 1 - \lambda_2$ denote the *eigenvalue gap* of P. The eigenvalue gap is directly related to the expansion of G [AM, Ta, Alo, SJ]. In particular, if G is an expander ϵ will be large.

Let $x_0, x_1, \ldots$ be the sequence of vertices visited by the random walk on G, where x_0 is chosen according to some distribution $\mathbf{q}$ on the vertices. Let $A \subseteq V$. Let χ_A denote the indicator function $\chi_A(x) = 1$ if $x \in A$, and 0 otherwise. Let $t_n := \chi_A(x_1) + \cdots + \chi_A(x_n)$, the number of visits to A in n random walk steps.

We introduce some special notation. Let $\frac{\mathbf{q}}{\sqrt{\pi}}$ denote the vector with entries $\frac{\mathbf{q}}{\sqrt{\pi}}(x) = \frac{\mathbf{q}(x)}{\sqrt{\pi(x)}}$, and let $N_{\mathbf{q}} = \|\frac{\mathbf{q}}{\sqrt{\pi}}\|_2$. Let $\mathbf{1} = (11\cdots1)$ be the vector of all 1's. Logarithms are in base e unless otherwise subscripted.

We now state our main result, a large deviation bound for a random walk on a weighted graph, in terms of the eigenvalue gap.

THEOREM 2.1. *Consider the random walk on a weighted graph $G = (V, E)$ with initial distribution $\mathbf{q}$. Let $A \subseteq V$. Let t_n be the number of visits to A in n steps. For any $\gamma \geq 0$,*

$$\Pr[t_n - n\pi(A) \geq \gamma] \leq (1 + \gamma\epsilon/10n)\, N_{\mathbf{q}} e^{-\gamma^2\epsilon/20n}. \tag{2.1}$$

Remarks. (i) We may write $\mathrm{E}_\pi \chi_A$ for $\pi(A)$ in Theorem 2.1. As we will see, the proof of this theorem uses only that χ_A is a nonnegative function on the vertices of G such that $\|\chi_A\|_\infty \leq 1$. For an arbitrary nonnegative function f on the vertices of G, we may substitute f for χ_A in the definition of t_n. Equation (2.1) becomes

$$\Pr[t_n - n\mathrm{E}_\pi f \geq \gamma] \leq (1 + \gamma\epsilon/10n)\, N_{\mathbf{q}} e^{-(\gamma/\|f\|_\infty)^2\epsilon/20n}. \tag{2.2}$$

(ii) Applying the theorem to $G \setminus A$ gives the same bound on $\Pr[t_n - n\pi(A) \leq -\gamma]$.

Before proving Theorem 2.1, we lay the groundwork for our proof with Theorem 2.2, a simplified version of a result of Höglund [H, Theorem 5.5]. This result follows the strategy of Cramér [Cr] and Chernoff [Ch], which is to estimate the deviation probability in terms of the *moment generating function* $m(r) = \mathrm{E}e^{rt_n}$, evaluated at some $r > 0$. We will see that this strategy reduces the problem of estimating the left-hand side of (2.1) to a problem of analyzing a perturbation of the transition matrix P.

Let $P(r) = PE_r$, where $E_r = \mathrm{diag}(e^{r\chi_A})$ and r is any complex number (we will often restrict r to the nonnegative real line). $P(r)$ is equal to P except that for $j \in A$ the jth column vector of P is multiplied by e^r.

P and $P(r)$ are similar to symmetric matrices. Let M be the (symmetric) weighted adjacency matrix of G: the ijth entry of M is w_{ij} if $\{i, j\} \in \mathrm{E}$ and 0 otherwise. Let $D = \mathrm{diag}(1/w_i)$. Then

$$\begin{aligned} P &= \sqrt{D} S \sqrt{D^{-1}}, \text{ and} \\ P(r) &= \sqrt{DE_r^{-1}}\, S(r) \sqrt{E_r D^{-1}}, \end{aligned} \tag{2.3}$$

where $S := \sqrt{D} M \sqrt{D}$ and $S(r) := \sqrt{DE_r} M \sqrt{DE_r}$ are symmetric. By (2.3) the eigenvalues of $P(r)$ are real for $r \geq 0$, and they are equal to the eigenvalues of $S(r)$;

in this case let $\lambda(r)$ and $\lambda_2(r)$ denote the largest and second largest eigenvalues of $P(r)$, respectively. Note that $P(0) = P$, $\lambda(0) = 1$ (with left eigenvector π^T and right eigenvector $\mathbf{1}$), and $\lambda_2(0) = \lambda_2$. For $r \geq 0$, let the eigenvalue gap of $P(r)$ be denoted by $\epsilon_r = \lambda(r) - \lambda_2(r)$.

THEOREM 2.2. *Consider the random walk on a weighted graph $G = (V, E)$ with initial distribution $\mathbf{q}$. Let $A \subseteq V$. Let t_n be the number of visits to A in n steps. For any $\gamma \geq 0$ and $r \geq 0$,*

$$\Pr[t_n - n\pi(A) \geq \gamma] \leq e^{-r(n\pi(A)+\gamma) + n \log \lambda(r)} \, (\mathbf{q}\, P(r)^n \, \mathbf{1})/\lambda(r)^n \,. \tag{2.4}$$

Proof. By Markov's inequality,

$$\begin{aligned} \Pr[t_n \geq n\pi(A) + \gamma] &= \Pr[e^{rt_n} \geq e^{r(n\pi(A)+\gamma)}] \\ &\leq e^{-r(n\pi(A)+\gamma)} \mathrm{E}_{\mathbf{q}} e^{rt_n} \,, \end{aligned} \tag{2.5}$$

where $\mathrm{E}_{\mathbf{q}}$ denotes the expectation given that x_0 is chosen according to $\mathbf{q}$. This expectation can be evaluated by summing over all possible trajectories $x_0, x_1, \ldots, x_n$ (where t_n is understood to be a function of the trajectory):

$$\mathrm{E}_{\mathbf{q}} e^{rt_n} = \sum_{x_0,\ldots,x_n} e^{rt_n} \mathbf{q}(x_0) \prod_{i=1}^{n} p_{x_{i-1}x_i} = \mathbf{q}\, P(r)^n \, \mathbf{1} \,. \tag{2.6}$$

Combining (2.6) with inequality (2.5) we obtain

$$\Pr[t_n \geq n\pi(A) + \gamma] = e^{-r(n\pi(A)+\gamma) + n \log \lambda(r)} \, (\mathbf{q}\, P(r)^n \, \mathbf{1})/\lambda(r)^n. \tag{2.7}$$ □

Here are the reasons that this theorem is useful.

1. For matrices $P(r)$ satisfying (2.3), the fraction $(\mathbf{q}\, P(r)^n \, \mathbf{1})/\lambda(r)^n$ on the right-hand side of (2.4) is close to 1. In fact, this fraction turns out to measure how close the starting distribution $\mathbf{q}$ is to the stationary distribution π. We have the following lemma.

LEMMA 2.3. *For $0 \leq r \leq 1$, $(\mathbf{q}\, P(r)^n \, \mathbf{1})/\lambda(r)^n \leq (1 + r)\, N_{\mathbf{q}}$.*

2. The exponent $-r(n\pi(A) + \gamma) + n \log \lambda(r)$ in the right-hand side of (2.4) is negative for small r because, as we show below, $\frac{d \log \lambda}{dr}|_{r=0} = \pi(A)$, and because $\log \lambda(0) = 0$. Our goal then is to bound this exponent away from zero for some r in order that a meaningful bound on $\Pr[t_n - n\pi(A) \geq \gamma]$ will follow from Theorem 2.2. This is accomplished by Lemma 2.4.

LEMMA 2.4. *If r is a real number such that $0 \leq e^r - 1 \leq \epsilon/4$, then*

$$\log \lambda(r) \leq r\pi(A) + 5r^2/\epsilon \,.$$

Proof of Theorem 2.1. We combine (2.4), Lemma 2.3, and Lemma 2.4 to get

$$\Pr[t_n - n\pi(A) \geq \gamma] \leq (1 + r)\, N_{\mathbf{q}} \, e^{-n(r\gamma/n - 5r^2/\epsilon)} \,. \tag{2.8}$$

The expression $r\gamma/n - 5r^2/\epsilon$ is quadratic in r and is maximized when $r = \gamma\epsilon/10n$, which satisfies the condition of Lemma 2.4 that $e^r - 1 \leq \epsilon/4$ (we can assume $\gamma < n$, because otherwise Theorem 2.1 is trivially true). The maximum value is $\gamma^2\epsilon/20n^2$. Substituting into (2.8),

$$\Pr[t_n - n\pi(A) \geq \gamma] \leq (1 + \gamma\epsilon/10n)\, N_{\mathbf{q}} \, e^{-\gamma^2\epsilon/20n}.$$

This completes the proof. □

Proof of Lemma 2.3. Since the matrix $S(r)$ of (2.3) is symmetric, $\|S(r)\|_2 = \lambda(r)$. We have

$$\begin{aligned}(\mathbf{q}\,P(r)^n\,\mathbf{1})/\lambda(r)^n &= (\mathbf{q}\,\sqrt{DE_r^{-1}}\,S(r)^n\,\sqrt{E_rD^{-1}}\,\mathbf{1})/\lambda(r)^n \\ &\le e^{r/2}\,(\,\|\tfrac{\mathbf{q}}{\sqrt{\pi}}\|_2\,\|S(r)^n\|_2\,\|\sqrt{\pi}\|_2\,)/\lambda(r)^n \\ &\le e^{r/2}\,N_{\mathbf{q}} \\ &\le (1+r)\,N_{\mathbf{q}}\,. \qquad \square\end{aligned}$$

In the remainder of this section we prove Lemma 2.4. For each $r \ge 0$ we define the matrix $B(r) = \frac{1}{e^{-1}}(S(r+1)-S(r))$. $B(r) \le S(r)$ in each entry. Also, $(B(r))_{i,j} = (S(r))_{i,j}$ whenever $i \in A$ and $j \in A$, and $(B(r))_{i,j} = 0$ whenever $i \notin A$ and $j \notin A$. Fix r, $0 \le e^r - 1 \le \epsilon/4$. We may expand the function $\log\lambda(y)$ in a Taylor series of the following form about the point $y = r$ (see [W]):

$$\log\lambda(y) = \log\lambda(r) + m_r(y-r) + (y-r)^2\int_0^1 (1-t)V_{r+(y-r)t}\,dt\,, \tag{2.9}$$

where m_z and V_z are the first and second derivatives, respectively, of $\log\lambda(y)$ at the point $y = z$. m_0 is equal to $\sqrt{\pi}B(0)\sqrt{\pi} = \pi(A)$, the limit of the mean of t_n/n as $n \to \infty$. This follows from letting $\mathbf{1}(r)$ be the right eigenvector of $P(r)$ with eigenvalue $\lambda(r)$ and equating coefficients in the power series expansions of both sides of $P(r)\mathbf{1}(r) = \lambda(r)\mathbf{1}(r)$ [W, p. 69]. V_0 is the limit of the variance of t_n/n as $n \to \infty$ [Nag, equation (2.5)].

The lemma follows from (2.9) and the second part of the following.

Claim 1. If $0 \le e^r - 1 \le \epsilon/4$, then

(i) $\epsilon_r \ge 3\epsilon/4$,

(ii) $V_r \le 10/\epsilon$.

Proof. For part (i), it is enough to show that $\epsilon_r \ge \epsilon - (e^r - 1)$. Note that for $r \ge 0$, the matrices $P(r)$ and $S(r)$ have the same eigenvalues, $S(r)$ is nonnegative, and $S(r) \ge S$ in each entry. By the Perron–Frobenius theorem, $\lambda(r) \ge 1$ [Se]. Let $\mu < \lambda(r)$ be any other eigenvalue of $S(r)$. It will suffice to show that $\mu \le \lambda_2 + e^r - 1$.

The matrix S is diagonalizable; there exist a unitary matrix U and diagonal matrices D' and D_A such that

$$\begin{aligned}B(0) \;&=\; 1/2(SD_A \,+\, D_AS)\,, \\ D' \;&=\; U^TSU, \quad \text{and} \\ \|D'\|_2 \;&=\; \|D_A\|_2 \;=\; 1\,.\end{aligned}$$

If $\mu \le \lambda_2$ we are done. If not, the matrix product $U^T(S + (e^r-1)B(0) - \mu I)U$, which is equal to $(D' - \mu I)[I + (D'-\mu I)^{-1}1/2(e^r-1)(D'U^TD_AU + U^TD_AUD')]$, is singular. Therefore,

$$\begin{aligned}1 &\le 1/2\|(D'-\mu I)^{-1}(e^r-1)D'U^TD_AU\|_2 \\ &\quad + 1/2\|(D'-\mu I)^{-1}(e^r-1)U^TD_AUD'\|_2 \\ &\le (e^r-1)/(\mu-\lambda_2)\,.\end{aligned}$$

(The first inequality uses the continuity of the function $\lambda_2(\cdot)$.) This proves part (i).

Our strategy for part (ii) is to use Cauchy's estimate from complex analysis to bound V_r in terms of the maximum value attained by $\lambda(z)$ in a complex neighborhood

of r [Ahl]. We bound this maximum value indirectly: the convergence of a certain loop integral will imply that $\lambda(z)$ lies inside the loop.

For z in a small complex neighborhood of r we may write $S(z) = S(r) + (e^{z-r} - 1)B(r)$. A fundamental theorem of perturbation theory says that the matrix for the projection onto the eigenspace for $\lambda(z)$ is given by the operator-valued complex integral

$$-\tfrac{1}{2\pi i} \int_\Gamma (S(z) - \zeta I)^{-1}\, d\zeta\,,$$

where Γ is any circle with $\lambda(r)$, and no other eigenvalues of $S(r)$, in its interior [Kat, section II.1.4]. The important fact for us is that if $\lambda(z) \in \Gamma$ then the integrand will have a singularity at $\lambda(z)$. To avoid this we choose Γ to have center $\lambda(r)$ and radius $\epsilon_r/2$. The norm of the integrand is finite on Γ as long as the following holds [Kat, section II.3.1]:

$$\begin{aligned} |e^{z-r} - 1| &< \|B(r)(S(r) - (\lambda(r) - \epsilon_r/2)I)^{-1}\|_2^{-1} \\ &\le 2/\epsilon_r\,. \end{aligned} \tag{2.10}$$

For the range of r we are interested in, in order for (2.10) to hold it is enough that

$$|z - r| \,<\, 3\epsilon_r/8\,. \tag{2.11}$$

Whenever (2.11) holds $\lambda(z)$ does not lie on Γ. But by continuity of $\lambda(z)$, (2.11) must imply that $\lambda(z)$ lies *inside* Γ, and therefore $|\lambda(z) - \lambda(r)| \le \frac{\epsilon_r}{2}$. Comparing Taylor series for $\lambda(z)$ and $\log \lambda(z)$, we see that $V_r \le 2\,\frac{d^2\lambda}{dz^2}|_{z=r}$. Cauchy's estimate says that $\frac{d^2\lambda}{dz^2}|_{z=r}$ is at most $(\max\{\lambda(z) : |z - r| < \rho\})/\rho^2$. If we let $\rho = \frac{3\epsilon_r}{8}$ then $V_r \le 2\frac{\epsilon_r}{2}/(\frac{3\epsilon_r}{8})^2 = 64/9\epsilon_r$. By part (i), $V_r \le 10/\epsilon$, and this completes the proof of Claim 1. □

Recent work of Dinwoodie [Di95a, Di95b] has improved the exponent in the bound of Theorem 2.1 by a factor of $20/\pi(A)$ for γ/n less than a small constant and has extended it to nonreversible Markov chains for γ/n dependent on the eigenvalue gap. This resolves an open question in the earlier version of this paper [G93b]. These results use perturbation theory and power series expansions of the perturbed matrix $P(r)$, eigenvalue $\lambda(r)$, and eigenvector $\mathbf{1}(r)$. These results also extend to estimates of $\mathrm{E}f$ for real-valued functions f on V. In [Kah], Kahale has also improved the exponent in the bound of Theorem 2.1. The exponent in the bound in [Kah] is given as the largest zero of a polynomial in λ_2, γ/n, and $\pi(A)$ and is shown to be optimal on two-state Markov chains for each triple of values of λ_2, γ/n, and $\pi(A)$. This result uses perturbation theory and a proof that a certain two-state Markov chain is extremal for the deviation probability being estimated.

3. (β, δ)-approximation algorithms for $\pi(A)$. In this section we compare different (β, δ)-approximation algorithms for $\pi(A)$ that use random walks to generate sample points from G. Formally, the output t of a (β, δ)-approximation algorithm for $\pi(A)$ must satisfy, with probability at least $1-\delta$, $\pi(A)(1-\beta) \le t \le \pi(A)(1+\beta)$. The cost of an algorithm will be the total number of random walk steps taken (see the discussion of measures of cost at the end of the section). We assume we can efficiently find one point s in G from which to start a random walk.

Below we define SP, AP, and a modified Aldous's procedure (APm). We analyze the efficiency of AP in Proposition 3.1. APm uses a technique for increasing the

confidence of estimates introduced by Jerrum, Valiant, and Vazirani in [JVV, Lemma 6.1]. This technique, the so-called "median trick," improves an arbitrary $(\beta, 1/4)$-approximation algorithm to a (β, δ)-approximation algorithm by taking $12\log(1/\delta)$ independent estimates and using the median of the estimates. We use Proposition 3.2 and Lemma 6.1 of [JVV] to analyze APm.

Standard procedure (SP). Start the random walk at s and simulate it for k' steps, so that the final state is distributed according to **q'**. Take the final state as a sample point. Repeat this l' times by choosing the same starting point s and taking a walk of length k' each time. Let $t_{l'}$ be the number of sample points in A, and let $t = t_{l'}/l'$.

Aldous's procedure (AP). Choose positive integers k and l. Start the random walk at s and simulate it for k steps (the "delay"), so that the final state x_0 is distributed according to **q**. Starting from x_0, continue the random walk l more steps taking each subsequent point as a sample point. Let t_l be the number of sample points in A, and let $t = t_l/l$.

Aldous's procedure, modified (APm). Choose k and l and follow AP to get an estimate of $\pi(A)$. Repeat $12\log(1/\delta)$ times and let t be the median of the estimates.

For a distribution **d** on the vertices of G, let the *chi-square distance from* π be defined by $\chi^2_d = \sum_x \pi(x)(\frac{\mathbf{d}(x)}{\pi(x)} - 1)^2$. Let χ^2_s denote the chi-square distance from π of the initial distribution concentrated at s.

PROPOSITION 3.1. *The cost of estimating* $\pi(A)$ *to within* $\beta\pi(A)$ *with probability* $1 - \delta$ *using AP is at most*

$$\log(1/\pi(s))\,/\,\epsilon \;+\; 20\log(8/\delta)\,/\,(\epsilon\beta^2\pi(A)^2)\,.$$

Proof. Let $k = \log(1/\pi(s))\,/\,\epsilon$ and $l = 20\log(8/\delta)\,/\,(\epsilon\beta^2\pi(A)^2)$. In the notation of Theorem 2.1, $N_{\mathbf{q}} = 1 + \chi^2_{\mathbf{q}}$. According to a result of Fill [Fi, equation (2.11)], $\chi^2_{\mathbf{q}} \le \chi^2_s(1-\epsilon)^k$. Therefore, $N_{\mathbf{q}} \le 1 + \chi^2_s(1-\epsilon)^k \le e^{-\epsilon k}/\pi(s) \le 2$.[2] By Theorem 2.1 and the ensuing remark (ii),

$$\Pr[|t_l/l - \pi(A)| \ge \beta\pi(A)] \;\le\; 4N_{\mathbf{q}}e^{-\beta^2\pi(A)^2\epsilon l/20} \;\le\; \delta. \qquad \square$$

Let $\pi_{\min} = \min_x \pi(x)$. Proposition 4.2 of Aldous [Ald87] and Lemma 6.1 of Jerrum, Valiant, and Vazirani [JVV] show that the cost of AP, modified, is $O[(\log(1/\delta)\,/\,\epsilon)(\log(1/\pi_{\min}) + 1\,/\,(\beta^2\pi(A)))]$. The following proposition and its corollary serve to replace $\pi_{\min}$ by $\pi(s)$. This will be useful in the applications of the next section.

PROPOSITION 3.2. *Let* $\beta > 0$ *and* $\alpha \le \beta^2$. *Let AP be used with parameters* $k = \log(20/(\pi(s)\alpha\pi(A)^2))/\epsilon$ *and* $l = 10/(\epsilon\beta^2\pi(A))$ *to generate an estimate* t. *Then*

$$|\mathrm{E}[t - \pi(A)]| \;\le\; \alpha\pi(A)^2/20$$

and

$$\mathrm{E}(t - \pi(A))^2 \;\le\; \beta^2\pi(A)^2/4\,.$$

[2] Fill's result actually depends on the larger of λ_2 and $|\lambda_{|V|}|$, but it is possible to modify any random walk so that in fact λ_2 is the larger of the two. One converts it to a so-called "lazy" random walk which with probability 1/2 stays put at each step [LS]. The values of k and l increase by at most a factor of 2. The same remark applies to the application of [JS91, Theorem 6.1] below.

TABLE 3.1
Costs of procedures for estimating $\pi(A)$.

Algorithm	Cost
SP	$\frac{1}{\epsilon}\log\frac{1}{\delta}\,\log\frac{1}{\pi(s)}\,\frac{1}{\beta^2\pi(A)}$
AP	$\frac{1}{\epsilon}\,(\log\frac{1}{\pi(s)}\,+\,\log\frac{1}{\delta}\,\frac{1}{\beta^2\pi(A)^2})$
APm	$\frac{1}{\epsilon}\log\frac{1}{\delta}\,(\log\frac{1}{\pi(s)}\,+\,\frac{1}{\beta^2\pi(A)})$

Proof. Let $\|\mathbf{q}-\pi\|$ denote the total variation distance between $\mathbf{q}$ and π. Citing, for example, [JS91, Theorem 6], we have that $\|\mathbf{q}-\pi\| < \frac{1}{20}\alpha\pi(A)^2$. Let $x_1,\ldots,x_l$ be the random vertices of G sampled. Each x_i is distributed according to some $\mathbf{q}_i$, which also satisfies $\|\mathbf{q}_i-\pi\| < \frac{1}{20}\alpha\pi(A)^2$. By linearity of expectations,

$$|\mathrm{E}[t_l/l-\pi(A)]| \le \frac{1}{l}\sum_{i=1}^{l}|\mathbf{q}_i(A)-\pi(A)| \le \frac{1}{20}\alpha\pi(A)^2.$$

Now define a vector $\mathbf{b}$ by letting $\mathbf{b}_j = \mathrm{E}[(t_l/l-\pi(A))^2 \,|\, x_0 = j]$. Observe that $\|\mathbf{b}\|_\infty \le 1$; therefore, by Proposition 4.1 of [Ald87], $\mathrm{E}_\pi\mathbf{b} \le 2\pi(A)/(\epsilon l)$. We have

$$\begin{aligned}\mathrm{E}(t-\pi(A))^2 &\le \mathrm{E}_{\mathbf{q}}\mathbf{b}\\ &\le [\mathrm{E}_\pi\mathbf{b} + |\mathrm{E}_{\mathbf{q}}\mathbf{b} - \mathrm{E}_\pi\mathbf{b}|]\\ &\le [2\pi(A)/(\epsilon l) + \alpha\pi(A)^2/20]\\ &\le \beta^2\pi(A)^2/4\,. \qquad \square\end{aligned}$$

COROLLARY 3.3. *The cost of estimating $\pi(A)$ to within $\beta\pi(A)$ with probability $1-\delta$ using APm is at most*

$$12\log(1/\delta)[\log(20/(\pi(s)\alpha\pi(A)^2))/\epsilon + 10/(\epsilon\beta^2\pi(A))]\,.$$

Proof. Set k and l as in Proposition 3.2 with $\alpha = \beta^2$. By Chebyshev's inequality,

$$\begin{aligned}\Pr[|t-\pi(A)| \ge \beta\pi(A)] &\le (\,\mathrm{E}(t-\pi(A))^2\,)/(\beta^2\pi(A)^2)\\ &\le 1/4.\end{aligned}$$

The corollary follows from Lemma 6.1 of [JVV]. $\square$

The standard procedure can be used to estimate $\pi(A)$ to within $\beta\pi(A)$ with probability $1-\delta$ by choosing $k' = O((\log(1/\pi_{\min}))\,/\,\epsilon)$ and $l' = O(\log(1/\delta)\,/\,(\beta^2\pi(A)))$. This value of k' comes from the analysis of Sinclair and Jerrum in [SJ], and the value of l' comes from Chernoff's bound. The method used in Lemma 3 of [JS91] is easily seen to be generally applicable; this improves the $\pi_{\min}$ to $\pi(s)$ in the expression for k'.

The costs of the three procedures disregarding constants are shown in Table 3.1. APm improves SP by a factor of $\min(\log(1/\pi(s)),\, 1/(\beta^2\pi(A)))$. AP is within a factor of $O(1/\pi(A))$ of APm. APm incurs extra cost by repeating the initial stage of finding a nearly random starting point; therefore, AP becomes better for small δ when β and A are such that $1/(\beta^2\pi(A)^2) < \log(1/\pi(s))$.

Remarks. (i) The recent work of Dinwoodie [Di95b] has improved the exponent in the bound of Theorem 2.1 by a factor of $20/\pi(A)$ for γ/n less than some small constant. This makes the cost of AP proportional to $1/\pi(A)$ and not $1/\pi(A)^2$, for β less than some small constant.

(ii) The results of this section all have analogues for estimating the expectation $\mathrm{E}f$ of a nonnegative function f on the vertices of G. Corresponding to Proposition 3.2 is the following, which we state without proof.

PROPOSITION 3.4. *Let $\beta > 0$ and $\alpha \le \beta^2$. Let $k = \log(20/(\alpha\pi(s)(\mathrm{E}f)^2))/\epsilon$ and $l = 10\mathrm{Var}(f)/(\epsilon\beta^2(\mathrm{E}f)^2)$. Take a random walk on G starting in s, and let x_i, $i \ge 1$, stand for the $(i+k)$th vertex of the random walk. Let $t = \frac{1}{l}(f(x_1) + \cdots + f(x_l))$. Then*

$$|\mathrm{E}[t - \mathrm{E}f]| \le \alpha(\mathrm{E}f)^2/20$$

and

$$\mathrm{E}(t - \mathrm{E}f)^2 \le \beta^2(\mathrm{E}f)^2/4\,.$$

The running time for AP is proportional to $\|f\|_\infty/(\mathrm{E}f)^2$ instead of $1/\pi(A)^2$, and the running time of SP has the same dependence as APm on $\mathrm{Var}(f)$ and $(\mathrm{E}f)^2$ [JS91, Lemma 3].

(iii) *Measures of cost.* It can be argued that SP has the advantage of taking only a small fraction (around ϵ) of the number of sample points of either AP or APm. We have not considered the number of sample points in our measure of cost because in the cases we know of the cost of sampling a point is dominated by the cost of taking one step of the random walk. For example, in the case of the random walk on matchings treated below, to determine the transition probability from one matching to another it is necessary to know whether the number of edges is going up or down by one, or staying the same. Therefore, it adds only a constant cost to keep track of the size of the current matching (see [JS89]). The same sort of justification holds for the case of the Ising model considered below. Computing the value of f at a vertex is no more difficult than computing the next transition probability (see [JS91]). Situations may yet arise of having to estimate $\mathrm{E}f$ for complicated f, where the number of sample points will be an important part of the cost of an algorithm.

(iv) *Nondelayed samples.* Suppose instead of generating a random initial point we had begun sampling immediately from s. The question is raised in [Ald87, Example 4.2] whether such nondelayed samples give good estimates in polynomial time. Setting $k = 0$ in Proposition 3.1 yields $l = 20(\log(1/\pi(s)) + \log(1/\delta))/(\epsilon\beta^2\pi(A)^2)$. An upper bound on $\log(1/\pi(s))$ is $\log|G| + \log\max_{x,y}(\pi(x)/\pi(y))$, which is typically polynomial in the data. This shows that nondelayed samples give good estimates in polynomial time, as long as $\pi(A)$ is not too small. It is not hard to see that the most efficient estimate, using our analysis, comes from letting k be equal to the relaxation time as in the proof of Proposition 3.1.

4. Applications: Two approximation algorithms.

The subgraphs random walk for the Ising model. In this section we give a modified version of an algorithm, due to Jerrum and Sinclair, for computing the partition function Z of a ferromagnetic Ising system I on m points (definitions below). Our new version of the algorithm improves the running time from $O(|E|^3m^7)$ to $O(|E|^2m^6)$, where m is the number of vertices and E is the edge set of the system [JS91].

Our algorithm is a *fully polynomial randomized approximation scheme* (*fpras*) as described in [KL] and in subsequent work on approximation algorithms. That is, on input I and a real number $a > 0$ it runs in time polynomial in m and $1/a$ and outputs a number that with probability at least $3/4$ approximates Z with relative error of a. The algorithm of Jerrum and Sinclair was the first fpras for this problem.

Consider a graph $I = ([m], E)$, $[m] = \{1, 2, \ldots, m\}$, with a positive *interaction energy* V_{ij} associated to each edge $\{i, j\} \in E$. A *configuration* $\sigma = \{\sigma_i\}_{i=1}^m$ is an assignment of positive ($\sigma_i = +1$) or negative ($\sigma_i = -1$) *spin* to each *site* $i \in [m]$. The *energy* of a configuration is given by the Hamiltonian

$$H(\sigma) = -\sum_{\{i,j\}\in E} V_{ij}\sigma_i\sigma_j - B\sum_{k\in[m]} \sigma_k,$$

where B is an external field.

This is a *ferromagnetic Ising system*; the positivity constraint on the V_{ij} models the behavior of a ferromagnet. A central problem for Bayesian inference in statistical physics is to compute a probability normalizing constant called the *partition function*:

$$Z = Z(V_{ij}, B, \beta_T) := \sum_{\sigma\in\{-1,+1\}^m} e^{-\beta_T H(\sigma)},$$

where β_T is related to the temperature. For further motivation we refer to [JS91] and the references therein.

Physicists have long used random walks on the set of spin configurations to estimate Z, but these random walks are not rapidly mixing for certain values of β_T. However, the partition function has an alternate characterization. Consider the set of all subsets X of the edge set E. Each subset X can be identified with a subgraph of I (which may contain isolated vertices). Let $\mathrm{odd}(X)$ stand for the set of all odd-degree vertices in the subgraph identified with X. Let $\lambda_{ij} = \tanh \beta_T V_{ij}$ and $\mu = \tanh \beta_T B$. The *weight* of X is defined by

$$w(X) = \mu^{|\mathrm{odd}(X)|} \prod_{\{i,j\}\in X} \lambda_{ij}. \tag{4.1}$$

The "subgraphs-world" partition function is

$$Z' = \sum_{X\subseteq E} w(X). \tag{4.2}$$

The two partition functions are related by the formula

$$Z = AZ', \tag{4.3}$$

where $A = (2\cosh \beta_T B)^m \prod_{\{ij\}\in E} \cosh \beta_T V_{ij}$ (see [NM]).

Although the subgraphs have no physical meaning, it is natural in light of (4.3) to compute Z indirectly by computing Z'. We define a random walk on the set G of all subsets $X \subseteq E$. Fix $\mu = \mu(\beta_T)$. For each X the transition probabilities out of X are given by the following rules [JS91, p. 16]: pick an edge $e \in E$ uniformly at random, and then

1. set $Y = X \oplus e$ (the symmetric difference of X and e);
2. if $w(Y) \geq w(X)$ then move to Y with probability 1;

(1) $A := (2\cosh\beta_T B)^m \prod_{\{ij\}\in E} \cosh\beta_T V_{ij}$; and $Z'(1) := \prod_{\{i,j\}\in E}(1+\lambda_{ij})$; $\Pi := A \times Z'(1)$;
(2) $r :=$ the natural number satisfying $\frac{m-r}{m} > \tanh\beta_T B \geq \frac{m-r-1}{m}$;
(3) $\mu_0 := 1$; **for** $j = 0, 1, \ldots, r$, **do begin**
  (4) **if** $j \leq r$, **then** $\mu_{j+1} := \frac{m-j}{m}$; **else** $\mu_{j+1} := \tanh\beta_T B$;
  (5) Make a call to AP with parameters k and l and t as in Proposition 3.4: let $\alpha = a^2/2m^2$ and $\beta = a/\sqrt{2m}$, and assume a lower bound of $1/10$ on $\mathrm{E}f$. Use the random walk on G with parameter μ_j, and obtain an estimate t for $\mathrm{E}_{\mu_j} f$;
  (6) $\Pi := \Pi \times t$;
end;
(7) **halt** with output Π;

FIG. 4.1. *Algorithm for computing the partition function.*

3. if $w(Y) < w(X)$ then move to Y with probability $w(Y)/w(X)$, and stay at X otherwise.

It is easy to see that G is connected by the transitions that have positive probability. It is not hard to check that the unique stationary distribution $\pi = \pi^\mu$ is given by $\pi(X) = w(X)/Z'$ (the transition probabilities were chosen with this goal in mind, according to the well-known Metropolis rule).

We adhere to the strategy of Jerrum and Sinclair for computing $Z' = Z'(\mu)$. This is to notice that $Z'(1) = \prod_{\{i,j\}\in E}(1+\lambda_{ij})$, which is easy to compute. Then for a general $\mu \in [0,1]$, one bootstraps from $Z'(1)$ down to $Z'(\mu)$ by computing successive ratios $Z'(\mu')/Z'(\mu)$ as follows. Let $\mathrm{E}_\mu f$ be the expectation under π^μ. For $0 \leq \mu_1 < \mu_0 \leq 1$ define the function $f(X) = (\mu_1/\mu_0)^{|\mathrm{odd}(X)|}$. The expectation of f satisfies $\mathrm{E}_{\mu_0} f = Z'(\mu_1)/Z'(\mu_0)$ [JS91, p. 11]. Furthemore, Lemma 4 of [JS91] says that if $\mu_0 \leq \mu_1 + 1/m$, then $\mathrm{E}_{\mu_0} f \geq 1/10$.

Our $(a, 1/4)$ approximation scheme for computing the partition function is shown in Fig. 4.1. We let step (5) assume a lower bound of $\epsilon \geq (2m^4|E|^2)^{-1}$, as given by Sinclair in [Si]. This algorithm differs from the algorithm of Jerrum and Sinclair [JS89, p. 12] in two ways. First, we use AP rather than SP in step (5). Second, we choose $\beta = \theta(a/\sqrt{m})$ instead of $\beta = \theta(a/m)$ in step (5) and we observe that the accumulated errors in Π tend to cancel one another out. This observation has been used before by Dyer and Frieze. We formalize it in the following lemma, which essentially appears in [DF, page 10].

LEMMA 4.1. *Let $z_1, \ldots, z_m$ be independent nonnegative random variables with $\mathrm{E}z_i \leq \eta/2m^2$ for some $\eta \leq 1$, and $\mathrm{E}z_i^2 \leq \nu/m$ for some $\nu \leq 1$. Let $Y = \prod_{i=1}^m (1+z_i)$. Then $\mathrm{E}(Y-1)^2 \leq \eta/m + \nu + (\eta/m+\nu)^2$.*

Proof. Since $Y \geq 1$, $\mathrm{E}(Y-1)^2 = \mathrm{E}[Y^2 - 2Y + 1] \leq \mathrm{E}Y^2 - 1$. Therefore,

$$\begin{aligned}
\mathrm{E}(Y-1)^2 &\leq \mathrm{E}[\textstyle\prod_i (1+z_i)^2] - 1 \\
&= \textstyle\prod_i (1 + 2\mathrm{E}z_i + \mathrm{E}z_i^2) - 1 \\
&\leq (1 + \eta/m^2 + \nu/m)^m - 1 \\
&\leq e^{\eta/m+\nu} - 1 \\
&\leq \eta/m + \nu + (\eta/m+\nu)^2. \qquad \square
\end{aligned}$$

THEOREM 4.2. *Let $a \in (0,1)$. The output Π of our algorithm satisfies $\Pi(1-a) < Z < \Pi(1+a)$ with probability at least $3/4$. The running time is at most $500m^6|E|^2/a^2$ random walk steps.*

Remark. Theorem 2, Lemmas 3 and 4, and the remark at the end of section 4 of [JS91] give a bound of $O(m^7|E|^3/a^2)$ random walk steps to estimate Z with the algorithm of Jerrum and Sinclair.

Proof. Let t_j be the estimate of $\mathrm{E}_{\mu_j} f$ from the jth iteration of the **do** loop. Let $z_j = t_j/\mathrm{E}f - 1$. Let $Y = \prod_j (1 + z_j) = \Pi/Z$. By Proposition 3.4,

$$|\mathrm{E}_{\mu_j} z_j| \le a^2/40m^2$$

and

$$\mathrm{E}_{\mu_j} z_j^2 \le a^2/8m.$$

Assuming in the worst case that $r = m$, the z_j satisfy the assumptions of Lemma 4.1 with $\eta = a^2/20$ and $\nu = a^2/8$. By Chebyshev's inequality, $\Pr[|(\Pi/Z) - 1| \ge a] \le \mathrm{E}(Y-1)^2/a^2$; so by the lemma,

$$\Pr[|(\Pi/Z) - 1| \ge a] \le [a^2/20 + a^2/8 + (a^2/20 + a^2/8)^2]/a^2 \le 1/4\,. \qquad \square$$

Perfect matchings. In this subsection we summarize the improvements we have made to an approximation algorithm, due to Jerrum and Sinclair, for counting the number of perfect matchings in a graph $H = (V, E)$ on $2m$ vertices. A *matching* is a subset of E such that no two edges share a common endpoint. A *perfect matching* is a matching that contains m edges. Let M_j denote the set of matchings of size j in H.

In [Br] Broder gave the first fpras for counting perfect matchings in dense graphs. A full proof of the correctness of this algorithm awaited Jerrum and Sinclair, who also gave a faster algorithm in [JS89].

An upper bound $q(m) > |M_{m-1}|/\max(1, |M_m|)$ is assumed to be known for some fixed polynomial q. Thus H is said to be *q-amenable*. Jerrum and Sinclair have shown that all dense graphs are m^2-amenable and almost every random bipartite graph is m^{10}-amenable in the $B(n,p)$ model for any density p above the threshold for the existence of a perfect matching [JS89]. Kenyon, Randall, and Sinclair have shown that all bipartite Cayley graphs and all regular periodic lattices of any dimension are $O(m^2)$-amenable [KRS]. In general, $q(m) \ge m$.

Jerrum and Sinclair define a random walk on a weighted graph G whose vertex set is all matchings of H: $\bigcup_{1 \le j \le m} M_j$. The transition probabilities depend upon a certain real parameter c which varies at different stages in the algorithm. In each stage the transition probabilities are chosen according to the Metropolis rule to ensure that the stationary probabilities $\pi(M_j)$ are proportional to $c^j|M_j|$ for all j. See [JS89] for more details.

Our modified version of the algorithm of [JS89] also uses a random walk on G, and like the original algorithm, our version proceeds in $m - 1$ stages, each using a different value $c^{(j)}$ for c. Stage $j \ge 1$ calculates the ratio $|M_{j+1}|/|M_j|$. The algorithm multiplies the ratios together to obtain an estimate of $|M_m|$ ($|M_1|$ is simply the number of edges of H). Since each of these ratios satisfies $|M_{j+1}|/|M_j| = c\pi(M_{j+1})/\pi(M_j)$, the essential step of the algorithm is to estimate $\pi(M_{j+1})$ and $\pi(M_j)$.

Our version modifies the algorithm of Jerrum and Sinclair [JS89, Figure 2] in two ways. First, we use APm in our random walk calls to estimate the $\pi(M_j)$'s, instead of SP. Second, we introduce a new strategy for choosing the values of c which allows the algorithm to assume that both $\pi(M_{j+1})$ and $\pi(M_j)$ are $\Omega(1/m)$.

The idea behind the new strategy is as follows. By Theorem 5.1 of [JS89], the sequence $|M_1|, |M_2|, \ldots, |M_m|$ is log-concave. This means that $\log |E|^{-1} = \log(|M_0|/|M_1|)$

$\le \log(|M_1|/|M_2|) \le \cdots \le \log(|M_{m-1}|/|M_m|) \le \log q(m)$. Put $c_j = |M_{j-1}|/|M_j|$ for each j, and $c_{m+1} = q(m)$. Whenever $c \in [c_j, c_{j+1}]$, $\pi(M_j) \ge \pi(M_i)$ for all i. In particular, when $c = c_{j+1}$, $\pi(M_j)$ and $\pi(M_{j+1})$ are each at least $1/m$. Since all of our sampling procedures are more efficient when sampling large sets, we would like to choose $c = c_{j+1}$ at stage j of the algorithm. What our algorithm does is to increase c by a constant factor at each stage, so that $\log c$ is incremented through the range $[\log |E|^{-1}, \log q(m)]$ in constant-size steps. However, the algorithm also detects those areas of this range where the $\log(\pi(M_j)/\pi(M_{j+1}))$'s are clumped together, and in those areas it takes smaller steps. In the end the algorithm needs to try only $m + O(\log m)$ different values of c to ensure that each c_{j+1} is approximated well by one of them.

THEOREM 4.3. *Let $a \in (0, 1/2]$. If H is q-amenable then the output Π of our algorithm satisfies $\Pi(1-a) < M_m < \Pi(1+a)$ with probability at least $3/4$. Our algorithm uses $O(q^2 m^5 |E| \log m/a^2)$ random walk steps.*

Remarks. (i) The algorithm of Jerrum and Sinclair uses $O(q^3 m^6 |E| \log^2 m/a^2)$ random walk steps, assuming the lower bound on the eigenvalue gap given later by Sinclair in [Si].

(ii) We have tried to reduce the running time of our algorithm by another factor of $O(m)$ using Lemma 4.1. The problem we face is that we need very reliable estimates of each $c^{(j)}$ (as did Jerrum and Sinclair) to make sure that M_j and M_{j+1} would be large enough sets (in stationary probability) to sample from. The cost of these reliable estimates dominates the cost of each stage of the algorithm.

5. Entropy of sources. We now consider the special case of a random walk on an unweighted undirected graph G. We view the random walk as an information source whose alphabet is the vertex set V of G. Theorem 2.1 enables us to quantify the rate of convergence of the classical Shannon–McMillan asymptotic equipartition property for this information source. We will then be able to estimate the entropy of the source very quickly, in $O(\log |G|)$ steps when G has constant expansion and constant nonuniformity (defined below). For purposes of fixed-length noiseless coding of the source, this result will imply a bound on the error exponent in terms of the expansion of G.

Let G have eigenvalue gap ϵ. Define the *nonuniformity* ν of G by

$$\nu = \max_{x,y} \pi(x)/\pi(y).$$

Let M be twice the number of edges of G. Let $P = (p_{ij})$ be the transition matrix for the random walk on G. For all i, j, $p_{ij} = 1/d_i$, where d_i is the degree of vertex i. $\pi(i) = d_i / \sum_j d_j = d_i/M$; therefore, $M \ge \max_i 1/\pi(i) \ge \nu$. Let $x_0, x_1, \ldots$ be a random walk on G starting from the stationary distribution. Consider the random sequence $X = x_1, x_2, \ldots$ an information source.

The random walk as an information source. In general, the *entropy* $H(Y)$ of an information source $Y = y_1, y_2, \ldots$ is defined in terms of the ordinary Shannon entropy: $H(Y) = \lim_{n\to\infty} \mathrm{E}\, H[y_n | y_1, \ldots, y_{n-1}]$. In our case, in which there is an underlying stationary random walk on an unweighted graph, there is a simple formula:

$$\begin{aligned} H(X) = \mathrm{E}\, H(x_1|x_0) &= -\mathrm{E} \log_2 p_{x_0 x_1} \\ &= \mathrm{E}_\pi \log_2 d_{x_0}\,. \end{aligned} \tag{5.1}$$

The Shannon–McMillan theorem [Ash, Theorem 6.6.1] states that under an ergodicity assumption which is satisfied here, an information source Y has the *asymp-*

totic equipartition property (AEP): for a fixed length n of source sequences, there are asymptotically $2^{nH(Y)}$ source sequences each of asymptotic probability $2^{-nH(Y)}$, and the probability of the "bad" set of remaining sequences tends to zero as n tends to infinity. The next theorem establishes an upper bound on the probability of the "bad" set.

Consider the particular information source X and a finite sequence $x_1, \ldots, x_n$, generated by X. Define the *empirical entropy* V_n of this finite sequence by $V_n = -\frac{1}{n} \log_2 \Pr[x_1, \ldots, x_n]$.

THEOREM 5.1. *Let X be the information source generated by the random walk on G. Let $x_1, \ldots, x_n$ be a finite sequence generated by X, with empirical entropy V_n. Then*

$$\begin{aligned} &\Pr[|V_n - H(X)| \ \geq \ \gamma] \\ &\qquad \leq \ 4e^{-(\gamma - (\log_2 M)/n)^2 n\epsilon/20\log_2{}^2 \nu}. \end{aligned} \tag{5.2}$$

Proof. Define a nonnegative function g on the vertices of G by $g(x) = \log_2 d_x$. Set $f(x) = g(x) - \min_y g(y)$. Then $\|f\|_\infty = \log_2 \nu$ and by (5.1), $\mathrm{E}_\pi f = H(X) - \min_y g(y)$. Let $t_n = f(x_1) + \cdots + f(x_n)$. Then $V_n - \min_y g(y) = t_n/n + (\log_2 1/\pi(x_1))/n - (\log_2 d_{x_n})/n$. Using (2.2) and remark (i) following Theorem 2.1,

$$\begin{aligned} &\Pr[V_n - H(X) \ \geq \ \gamma] \\ &\quad = \ \Pr[t_n/n - (H(X) - \min_y g(y)) \\ &\qquad \geq \ \gamma - (\log_2 1/\pi(x_1))/n + (\log_2 d_{x_n})/n] \\ &\quad \leq \ 2\exp\left[-\left(\frac{\gamma - (\log_2 M)/n}{(\log_2 \nu)/n}\right)^2 \epsilon/20n\right] \\ &\quad \leq \ 2e^{-(\gamma - (\log_2 M)/n)^2 n\epsilon/20\log_2{}^2 \nu}. \end{aligned}$$

The inequality in the other direction is similar. □

Unifilar sources. Let $\chi : V(G) \to \{0, 1, \ldots, k-1\}$ be any labelling of the vertices of G such that the random walk on G is unifilar; that is, no vertex G is adjacent to two or more vertices of the same label. Let $Y = y_1, y_2, \ldots$ be the sequence of labels: $y_i = \chi(x_i)$. Then the entropy of the information source Y still satisfies (5.1): $H(Y) = \mathrm{E}_\pi \log_2 d_{x_0}$. The definition of empirical entropy for Y is also the same, and we immediately have the following generalization.

COROLLARY 5.2. *Let V_n denote the empirical entropy of $y_1, \ldots, y_n$. Then*

$$\begin{aligned} &\Pr[|V_n - H(Y)| \ \geq \ \gamma] \\ &\qquad \leq \ 4e^{-(\gamma - (\log_2 k|G|)/n)^2 n\epsilon/20\log_2{}^2 k}. \end{aligned} \tag{5.3}$$

□

Remarks. (i) The logarithm of the right-hand side of (5.3) gives an upper bound on the error exponent for fixed-length coding of the source Y. This appears to be the first such bound which applies for all n and is given in terms of a computable quantity (the eigenvalue gap). Large deviation methods were used in [Nat, An], giving asymptotic estimates of the error exponent.

(ii) Corollary 5.2 can be restated as follows: to estimate $H(Y)$ to within an additive error γ with probability at least $1 - \delta$ requires a random walk of length $\mathrm{O}(\log^2 k \log(1/\delta) \log k|G|/\epsilon\gamma^2)$. When the number of labels k is constant and G has constant expansion, this simplifies to $\mathrm{O}(\log(1/\delta) \log |G|/\gamma^2)$.

6. Further work. The main open problem is to give a lower bound on the probability of deviation estimated in Theorem 2.1. A lower bound must depend on the set A and not only on $\pi(A)$. For example, in the usual barbell graph, if A is all on one side of the "handle," the deviation probability will actually be larger than if A is equally distributed between the two sides of the graph [G93a, pp. 56–57].

It would be nice to be able to estimate the entropy of a random walk on a weighted graph. In [G93a], the author has shown that such a generalization would have applications to the problem of discriminating two hidden Markov chains from their output. The immediate difficulty is not the lack of a nice formula for entropy in this case, but the problem of estimating the expected value of a function on the edges of G. The random walk on G induces a Markov chain on the edges of G. The induced Markov chain is not time-reversible, so the Chernoff bound does not apply as it stands.

Ultimately it would be nice to be able to estimate the entropy of a `0-1` source generated by a (possibly nonunifilar) Markov chain whose states are labelled `0` and `1`.

Acknowledgments. My advisor Mike Sipser helped to formulate the problem of a central limit theorem for expander graphs and continually discussed the issues of this paper with me while my work progressed. Johan Håstad drew my attention to the paper of T. Höglund. László Lovász pointed out the potential application of a Chernoff bound to approximation algorithms. Nabil Kahale, Miklos Simonovits, Alistair Sinclair, and David Zuckerman, and Svante Janson made helpful comments. It is my pleasure to thank these people.

REFERENCES

[Ahl] L. Ahlfors, *Complex Analysis*, 2nd ed., McGraw-Hill, New York, 1966.

[AKS] M. Ajtai, J. Komlós, and E. Szemerédi, *Deterministic simulation of logspace*, in Proc. 19th ACM Symp. on Theory of Computing, 1987.

[Ald87] D. Aldous, *On the Markov chain simulation method for uniform combinatorial distributions and simulated annealing*, Probab. Engrg. Inform. Sci., 1 (1987), pp. 33–46.

[Alo] N. Alon, *Eigenvalues and expanders*, Combinatorica, 6 (1986), pp. 83–96.

[AM] N. Alon and V. D. Milman, λ_1*, isoperimetric inequalities for graphs, and superconcentrators*, J. Combin. Theory B, 38 (1985), pp. 73–88.

[An] V. Anantharam, *A large deviation approach to error exponents in source coding and hypothesis testing*, IEEE Trans. Inform. Theory, 36 (1990), pp. 938–943.

[Ash] R. Ash, *Information Theory*, Interscience, New York, 1965.

[Br] A. Z. Broder, *How hard is it to marry at random? (on the approximation of the permanent)*, in Proc. 18th ACM Symp. on Theory of Computing, 1986, pp. 50–58.

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

[Cr] H. Cramér, *Sur un nouveau théorème de la théorie des probabilités*, Actualités Scientifiques et Industrielles, 736 (1938).

[CW] A. Cohen and A. Wigderson, *Dispersers, deterministic amplification, and weak random sources*, in Proc. 30th IEEE Symp. on Foundations of Computer Science, 1989.

[Di95a] I. H. Dinwoodie, *A probability inequality for the occupation measure of a reversible Markov chain*, Ann. Appl. Probab., 5 (1995), pp. 37–43.

[Di95b] I. H. Dinwoodie, *Expectations for Nonreversible Markov Chains*, preprint, 1995.

[Do] J. L. Doob, *Stochastic Processes*, Wiley, New York, 1953.

[DF] M. Dyer and A. Frieze, *Computing the volume of convex bodies: A case where randomness provably helps*, in Proc. AMS Symp. on Probabilisitc Combinatorics and Its Application, 1991, pp. 123–170.

[DFK] M. Dyer, A. Frieze, and R. Kannan, *A random polynomial time algorithm for approximating the volume of convex bodies*, in Proc. 21st Annual ACM Symp. on Theory of Computing, 1989, pp. 375–381.

[Fi] J. A. Fill, *Eigenvalue bounds on convergence to stationarity for nonreversible Markov chains, with an application to the exclusion process*, Ann. Appl. Probab., 1 (1991), pp. 62–87.

[G93a] D. Gillman, *Hidden Markov Chains: Convergence Rates and the Complexity of Inference*, Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, 1993.

[G93b] D. Gillman, *A Chernoff bound for random walks on expander graphs*, in Proc. 34th IEEE Symp. on Foundations of Computer Science, 1993, pp. 680–691.

[G*] O. Goldreich, R. Impagliazzo, L. Levin, R. Venkatesan, and D. Zuckerman, *Security preserving amplification of hardness*, in Proc. 31st IEEE Symp. on Foundations of Computer Science, 1990, pp. 318–326.

[H] T. Höglund, *Central limit theorems and statistical inference for finite Markov chains*, Z. Wahr. Gebiete, 29 (1974), pp. 123–151.

[IZ] R. Impagliazzo and D. Zuckerman, *How to Recycle Random Bits*, in Proc. 30th IEEE Symp. on Foundations of Computer Science, 1989.

[JS89] M. Jerrum and A. Sinclair, *Approximating the permanent*, SIAM J. Comput., 18 (1989), pp. 1149–1178.

[JS91] M. Jerrum and A. Sinclair, *Polynomial-time approximation algorithms for the Ising model*, SIAM J. Comput., 22 (1993), pp. 1087–1116.

[JVV] M. Jerrum, L. Valiant, and V. Vazirani, *Random generation of combinatorial structures from a uniform distribution*, Theoret. Comput. Sci., 43 (1986), pp. 169–188.

[Kah] N. Kahale, *Large Deviation Bounds for Markov Chains*, DIMACS Technical Report, DIMACS, Rutgers University, New Brunswick, NJ, 1994, pp. 94–39.

[KL] R. Karp and M. Luby, *Monte Carlo algorithms for enumeration and reliablity problems*, in Proc. 15th ACM Symp. on Theory of Computing, 1983.

[Kat] T. Kato, *A Short Introduction to Perturbation Theory for Linear Operators*, Springer-Verlag, New York, 1982.

[KRS] C. Kenyon, D. Randall, and A. Sinclair, *Matchings in lattice graphs*, in Proc. 25th ACM Symp. on Theory of Computing, 1993, pp. 738–746.

[LS] L. Lovász and M. Simonovits, *Random walks in a convex body and an improved volume algorithm*, Random Structures Algorithms, 4 (1993), pp. 359–412.

[Nat] S. Natarajan, *Large deviations, hypothesis testing, and source coding for finite Markov chains*, IEEE Trans. Inform. Theory, 31 (1985), pp. 360–365.

[Nag] S. V. Nagaev, *More exact statements of limit theorems for homogeneous Markov chains*, Theory Probab. Appl., 6 (1961), pp. 62–81.

[NM] G. F. Newell and E. W. Montroll, *On the theory of the Ising model of ferromagnetism*, Rev. Modern Phys., 25 (1953), pp. 353–389.

[Se] E. Seneta, *Nonnegative Matrices*, Wiley, New York, 1973.

[Si] A. Sinclair, *Improved Bounds for Mixing Rates of Markov Chains and Multicommodity Flow*, Technical Report ECS-LFCS-91-178, Department of Computer Science, University of Edinburgh, Scotland, October 1991.

[SJ] A. Sinclair and M. Jerrum, *Approximate counting, uniform generation, and rapidly mixing Markov chains*, Inform. and Comput., 82 (1989), pp. 93–113.

[SS] G. W. Stewart and J.-G. Sun, *Matrix Perturbation Theory*, Academic Press, New York, 1990.

[Ta] R. M. Tanner, *Explicit construction of concentrators from generalized n-gons*, SIAM J. Algebraic Discrete Meth., 5 (1984), pp. 287–294.

[W] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, UK, 1965.

SIAM J. COMPUT.
Vol. 27, No. 5, pp. 1221–1236, October 1998
© 1998 Society for Industrial and Applied Mathematics
001

PROCESSOR-RING COMMUNICATION: A TIGHT ASYMPTOTIC BOUND ON PACKET WAITING TIMES*

E. G. COFFMAN, JR.†, NABIL KAHALE‡, AND F. T. LEIGHTON§

Abstract. We consider N processors communicating unidirectionally over a closed transmission channel, or ring. Each message is assembled into a fixed-length packet. Packets to be sent are generated at random times by the processors, and the transit times spent by packets on the ring are also random. Packets being forwarded, i.e., packets already on the ring, have priority over waiting packets. The objective of this paper is to analyze packet waiting times under a greedy policy within a discrete Markov model that retains the overall structure of a practical system but is simple enough so that explicit results can be proved. Independent, identical Bernoulli processes model message generation at the processors, and independently and identically distributed (i.i.d.) geometric random variables model the transit times. Our emphasis is on asymptotic behavior for large ring sizes, N, when the respective rate parameters have the scaling λ/N and μ/N. Our main result shows that, if the traffic intensity is fixed at $\rho = \lambda/\mu < 1$, then as $N \to \infty$ the expected time a message waits to be put on the ring is bounded by a constant. This result verifies that the expected waiting time under the greedy policy is within a constant factor of that under an optimal policy.

Key words. processor rings, processor interconnection networks, queuing networks, asymptotic analysis, routing algorithm analysis

AMS subject classifications. 68M20, 68R05, 90B12, 90B35

PII. S0097539794268637

1. Introduction.

Communication among N processors takes place counterclockwise along a slotted circular transmission channel, or *ring*. A processor generates messages, receives messages, and forwards messages between other processors. Each message is a *packet* of fixed duration. One time unit is required for a packet to be sent or forwarded from one processor to its counterclockwise neighbor. Packets are generated randomly at the processors according to i.i.d. arrival processes. The integer times spent by packets on the ring, packet *transit times*, are i.i.d. random variables. Packets being forwarded on the ring have priority; while a processor has a packet to be forwarded, it cannot place one of its own waiting packets on the ring. A packet waiting for transmission is held in a queue at the processor where it was generated.

The details defining a practical implementation of a processor ring are many and varied. Indeed, the applications and analysis of communication rings form a rather large and growing literature; see van Arem and van Doorn (1990), Barroso and Dubois (1993), and Georgiadis, Szpankowski, and Tassiulas (1997) for brief surveys and many references. As a concession to mathematical tractability, we adopt here the simple discrete Markov model in Fig. 1, where the ring is partitioned into *cells*, each capable of holding a single packet. The cells rotate counterclockwise past the processors in discrete steps, one step per unit of time. Packets are generated at each of the N

*Received by the editors May 19, 1994; accepted for publication (in revised form) June 17, 1996; published electronically May 19, 1998.

http://www.siam.org/journals/sicomp/27-5/26863.html

†Bell Laboratories, Lucent Technologies, Murray Hill, NJ 07974 (egc@bell-labs.com).

‡AT&T Research Laboratories, Florham Park, NJ 07932 (kahale@research.att.com). This research was partially supported by NSF contract STC-91-19999 and by the New Jersey Commission on Science and Technology while the author visited DIMACS, Rutgers University, New Brunswick, NJ.

§Deptartment of Mathematics and Laboratory for Computer Science, MIT, Cambridge, MA 02139 (ftl@math.mit.edu).

processors by a Bernoulli process at rate λ/N, $0 < \lambda < N$, per time unit (step); the total arrival rate is then λ. The packet transit times are geometrically distributed with rate parameter μ/N, $N > \mu > \lambda$. Thus, at any given step, a packet on the ring departs with probability μ/N and stays for at least one more step with probability $1 - \mu/N$, independent of how long the packet has already been on the ring. We will explain shortly the reason for the scaling of arrival and transit-time parameters by the ring size.

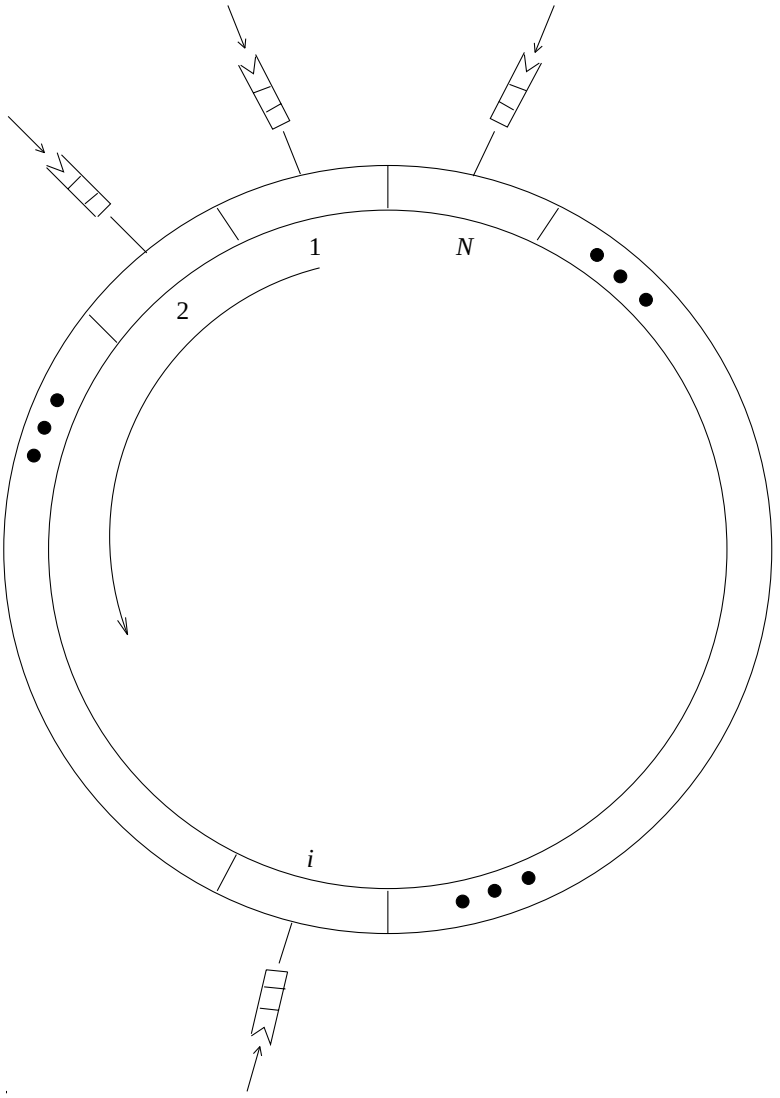


FIG. 1. *The rotating ring model.*

In each step, the ring system undergoes a transition according to the following sequence.

(i) The ring rotates one position while processor queues accept new arrivals, if any (at most one per queue in each step).

(ii) Packets on the ring that have completed their transit times are delivered, i.e., removed from their cells.

(iii) Each processor with a nonempty queue opposite an empty cell then puts a waiting packet into this cell.

This gives the *nonblocking* model; reversing (ii) and (iii) would give the *blocking* model: a departing packet can not be replaced in the same time step by a waiting packet. As we shall see, our asymptotic results apply to both models. The above sequence gives the *greedy* cell admission policy, placing waiting packets on the ring as soon as empty cells are available.

As discussed in Coffman et al. (1995), the greedy policy has the undesirable effect of occasionally "freezing out" certain processor queues for long periods of time; long trains of occupied cells pass by such processors, denying them access to the ring. The results of this paper will show that, for large rings within our probability model, the greedy rule is remarkably efficient, and that in fact, the above behavior is quite rare.

Our specific objective is to analyze packet waiting times under the greedy policy. (Hereafter, unless noted otherwise, waiting times always refer to times spent waiting in processor queues.) To prepare for the statement of our main theorem on waiting times, we need a little more notation. For a given admission policy, we denote the joint queue length at integer time t by $\mathbf{Q}(t) = (Q_1(t), \ldots, Q_N(t))$, where $Q_i(t)$ is the number in the ith processor queue at time t. The phrase "at time t" means at an instant just after t so that events, if any, occurring at t have already taken place. Define the N-bit vector $\mathbf{R}(t)$ whose ith bit is 1 if and only if a packet is in the ith cell at time t. Hereafter, the term *state* refers to a pair $\mathbf{Q}(t), \mathbf{R}(t)$ at some time t. It follows from the geometric law for transit times that the *ring process* $\{(\mathbf{Q}(t), \mathbf{R}(t)), t = 0, 1, \ldots\}$ is a Markov chain. It was shown by Coffman et al. (1995) that, if $\lambda < \mu$, then the ring process under the greedy rule is ergodic. Unfortunately, an exact analysis of the stationary behavior of this ring process seems quite difficult. Indeed, attempts to solve the balance equations have so far failed even for the case $N = 2$. Thus, we turn to asymptotic estimates for large ring sizes, N, with λ and μ fixed and $\lambda < \mu$. That is why we introduced the scalings λ/N and μ/N; as we allow N to increase, the traffic intensity will remain fixed at $\rho = \lambda/\mu$, the usual product of arrival rate and average service (transit) time.

With $\lambda < \mu$, let Q have the stationary distribution common to all queue lengths $Q_i(t)$, and let W be the waiting time of a packet in the stationary regime.

THEOREM 1.1. *Fix λ and μ with $\lambda < \mu$. Then, under the greedy policy,*

$$E[Q] = \Theta(1/N) \ .$$

Thus, by Little's theorem,

$$E[W] = \Theta(1) \ .$$

The lower bounds are easy to see, as follows. Consider the entire ring as an N-server system with a total arrival rate λ and maximum departure rate μ. Then, by Little's theorem, the arrival rate λ times the average time spent on the ring, i.e., N/μ, must be equal to the expected number of packets on the ring in the stationary regime, i.e., ρN. But if a positive fraction $\rho > 0$ of the ring is occupied on average, then there must be a positive average waiting time $E[W] = \Omega(1)$ to get on the ring and hence $E[Q] = \Omega(1/N)$.

In the usual way, *on-line* admission policies are those deciding packet admissions solely on the basis of information currently available about packets already in the system, waiting or on the ring. Such information can include, for example, queue

lengths and the elapsed times already spent in the system by packets. As we will see later, it is convenient to extend this class of admission policies by allowing decisions to depend also on the times and queues of future arrivals. *Hereafter, unless stated otherwise, the term* policy *refers to a policy in this extended class.* Note, in particular, that policies retain the on-line property with respect to transit times on the ring; i.e., we do not allow policies that base decisions on prior knowledge of remaining transit times.

We say that a policy A is optimal if, over any interval $[0, T]$, the sum of waiting times (in queue) under A is stochastically no larger than that under any other policy starting in the same initial state. We prove in the next section that the greedy policy is an optimal policy (the proof will need the geometric law for transit times). In the proof of Theorem 1.1, this result allows us to analyze a more tractable policy with the same asymptotic performance as the greedy rule; the more tractable policy exploits the fact that policies can base decisions on the times and queues of future arrivals.

Coffman et al. (1995) proved in an earlier paper that the growth of the expected waiting time in our model was sublinear in N, i.e., $E[W] = o(N)$. Our much stronger result shows that the expected waiting time is in fact bounded by a constant. So, by Little's theorem, an important practical implication of our result is that the expected size of a buffer needed to hold all waiting packets is bounded by a constant uniformly in N. The proof of Theorem 1.1 requires a much more intricate probabilistic analysis than the one in Coffman et al. (1995), where the law of large numbers was the basic tool. Here, we will need more powerful asymptotic bounds (e.g., those of Chernoff type) on the tail probabilities for sums of independent random variables and the excursions of Lindley processes (see, e.g., Prabhu (1965), p. 66); these appear as lemmas in section 3. The proof of the upper bound $E[Q] = O(1/N)$ is given in sections 4 and 5. The paper concludes in section 6 with a brief discussion of extensions and open problems.

2. Preliminaries. Consider the packet at the head of any given nonempty queue. Since transit times are geometrically distributed with parameter μ/N, the probability that this packet is placed on the ring in the current time step is at least μ/N; the conditional probability is precisely μ/N if the cell is occupied on arrival, and it is trivially 1 if the cell is empty. Thus, one expects that, in statistical equilibrium, the ith queue length Q_i is bounded stochastically for each i by the length of a single-server Markov (i.e., M/M/1) queue in discrete time with arrival and service rate parameters λ/N and μ/N. Moreover, this bound should hold independently for each queue. Indeed, these observations are but a special case of Theorem 2 in Coffman et al. (1995). An easy analysis of the discrete-time M/M/1 queue then proves the following lemma.

LEMMA 2.1. *For each i independently, Q_i is stochastically smaller than a nonnegative integer random variable L with $P(L = n) \sim (1 - \rho)\rho^n$ as $N \to \infty$ for every $n \geq 0$, and with*

$$P(L > n) = O(e^{-\nu n}) \ , \tag{2.1}$$

where $\nu = \ln 1/\rho > 0$.

Hereafter, we take the equivalent point of view that *the queues rotate past the ring of cells, which remains fixed.* As shown in Fig. 2, in any given time interval $[0, T]$, the ring process can be represented by events on a cylindrical lattice cut at some cell position and laid out as a rectangle. For simplicity, we assume that the cylinder is cut between cell N and cell 1. Along the top of the rectangle the $Q_i(0)$, $1 \leq i \leq N$,

give the initial state of the queues, and the bullets (•'s) indicate the initial cell states: a cell with a • at time 0 is empty; otherwise, it is occupied. Again, for simplicity, we assume queue 1 is at cell 1 at time 0. Within the rectangle, circles (∘'s) and bullets give a random sample of arrivals and departures, respectively. A • and ∘ can appear at the same lattice point. The probability of such an event is $O(1/N^2)$, and hence relatively low; for simplicity, the figures in this paper do not show samples where such coincidences occur.

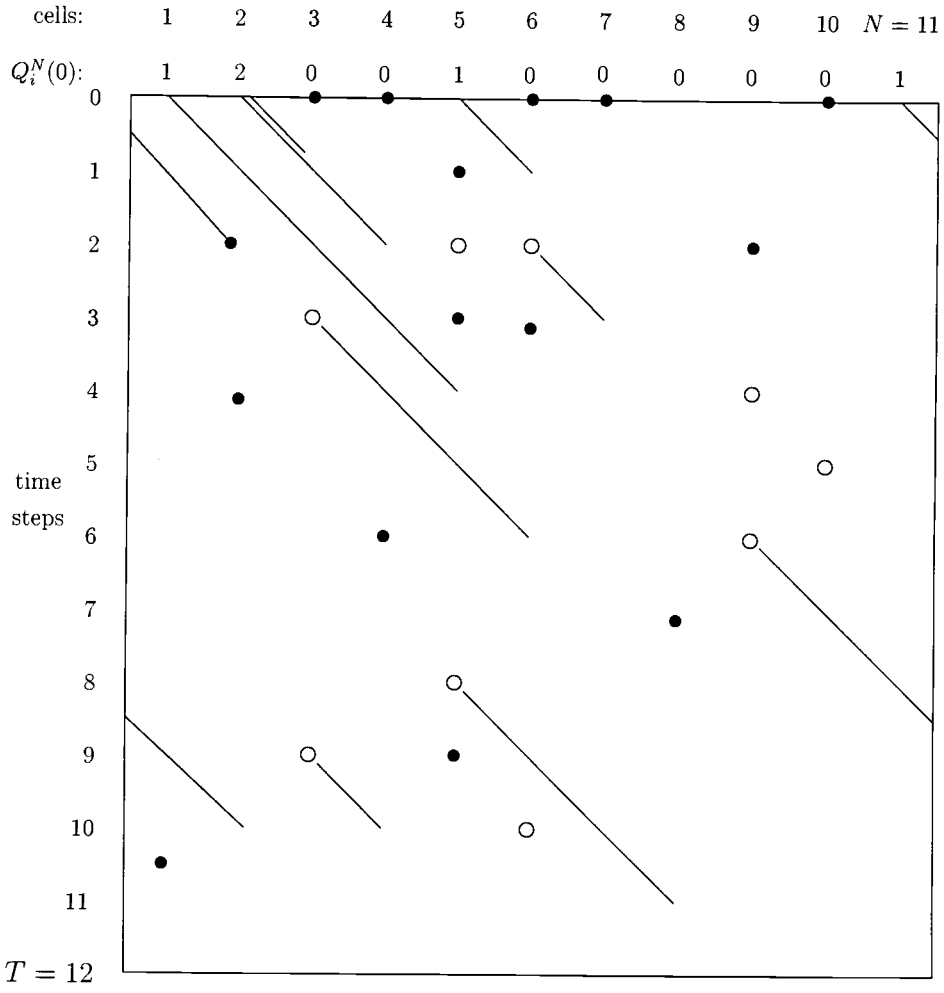


FIG. 2. *Greedy assignment.*

The greedy policy is represented by a suitable assignment of cells to circles (new arrivals) and to packets in the initial state. An example is shown in Fig. 2. The *motion* lines drawn between packets and assigned cells describe the trajectories of the packets in time and space; their vertical components correspond to waiting times. A motion line is broken into two pieces when it extends past cell N, one ending at the right boundary and one beginning at the same time at the left boundary.

To ensure that an assignment of circles and initial packets to cells is valid, one must check to see that the cell, say c, at which a motion line terminates, at time t say, is indeed empty at time t. Thus, if t', $0 \le t' < t$, is the time of the last departure (bullet) in cell c, then no other motion lines can terminate at cell c in the interval $[t', t]$.

We conclude this section with a proof that greedy is optimal in that it minimizes stochastically the sum S of waiting times over any given interval $[0, T]$. The proof uses the following simple relation between S and the queue lengths $Q_i(t)$ during $[0, T]$:

$$S = \sum_{t=0}^{T} \sum_{i=1}^{N} Q_i(t) \ . \tag{2.2}$$

THEOREM 2.1. *The greedy rule is an optimal admission policy.*

Proof. Consider ring operation over an interval $[0, T]$, and let A be an arbitrary policy. To compare total waiting times in $[0, T]$ under A and greedy, both starting in the same initial state, we compare both to an intermediate algorithm A*, which is artificial in that it sometimes returns packets to queues before they have completed their transit times. Under A* an occupied queue places a packet into the first available empty cell, just as with the greedy policy. But suppose that, under A* at some time t, a cell c occupied by packet φ is in front of a queue having a packet φ' that, under A, would have been in c at time t. Then at time t, φ and φ' change places under A*; φ joins the queue and φ' enters the cell.

At any given queue, the admissions under A* and greedy implement the same deterministic rule except at times when A* exchanges a packet in the queue with the packet in the cell in front of the queue. *But such an exchange does not change the state (any queue length or the state of any cell) of the ring process*; from the point of view of the ring process, the exchange has the effect of doing nothing, which is just what the greedy rule would do in the same circumstances. Thus, if A* and greedy start in the same initial state, then the joint queue-length process over $[0, T]$, and hence by (2.2) the sum of waiting times over $[0, T]$, is stochastically the same under A* and greedy.

It remains to show that the sum of waiting times over $[0, T]$ under A is at least as large stochastically as it is under A*. In fact, we prove the stronger deterministic result: for a given initial state, a given sequence of arrivals over $[0, T]$, and a given sample of the remaining transit times of all packets in the system during $[0, T]$, the sum of waiting times under A is at least that under A*. To see this, note first that, although A* may put a packet φ on and off the ring several times, eventually one of three events will occur: φ will depart, T steps will have been taken, or φ will be in a queue when the cell assigned to it under A catches up to it. In the last case, A* places φ on the ring making an exchange, if needed, and leaves it there until it departs or T steps have been made. Thus, every packet during $[0, T]$ has moved along the ring under A* at least as far as it has moved under A, and so the sum of waiting times under A* is deterministically at most the sum under A. □

3. Probability bounds. The reader may wish to skip this section at first reading, referring back to it as needed while reading section 5. We begin with a useful Chernoff bound that combines Theorems A.12 and A.13 of Alon and Spencer (1991), pp. 237–238.

LEMMA 3.1. *Let $Z = Z_1 + \cdots + Z_n$, where the Z_i are independent Bernoulli random variables with $P(Z_i = 1) = p_i$, $P(Z_i = 0) = 1 - p_i$. Then for any $\epsilon > 0$, there exists a $\beta > 0$ such that*

$$P((1-\epsilon)E[Z] < Z < (1+\epsilon)E[Z]) = 1 - O(e^{-\beta E[Z]}) \ . \tag{3.1}$$

Next, we consider a Lindley process, starting at the origin and defined by (x^+ denotes the positive part of x)

$$\zeta_0 = 0, \quad \zeta_i = (\zeta_{i-1} + U_i)^+, \tag{3.2}$$

with $U_i = X_i - Y_i$, where $\{X_i\}$ and $\{Y_i\}$ are independent sequences of i.i.d. random variables. In our application, Y_i is an integer in $\{0, \ldots, K\}$ with K a given integer constant independent of N, and X_i is the number of arrivals of a rate-$a\lambda/N$ Bernoulli process in bN time steps, where a and b are constants independent of N. Thus, for large N, X_i is approximately Poisson distributed with mean $ab\lambda$. It is easy to check that X_i and hence U_i has an exponential tail probability; i.e., there exists a $\kappa > 0$ such that

$$P(U_i > x) \le P(X_i > x) = O(e^{-\kappa x}) \ . \tag{3.3}$$

The process $\{\zeta_i\}$ is said to have negative drift if $E[Y_i] > E[X_i]$ and hence $E[U_i] < 0$. The next result follows from standard theory (e.g., see Asmussen (1987)). Let the X_i and U_i be distributed as X and U, respectively.

LEMMA 3.2. *If $E[U] < 0$, then $E[\zeta_i]$ is bounded by a constant uniformly in $i \ge 0$. The distributions of the ζ_i converge in total variation to the distribution of a random variable ζ with moments of all orders.*

In addition to Lemma 3.2, we will need certain probability bounds on excursions of $\{\zeta_i\}$. These will be derived in terms of corresponding bounds for the unrestricted process

$$\xi_i = \xi_{i-1} + U_i, \quad i \ge 1 \ , \tag{3.4}$$

with the U_i defined as before, and with a given initial state ξ_0. Hereafter, we assume a negative drift $E[U] < 0$.

The probability bound on excursions of $\{\xi_i\}$ that we will use in the analysis of $\{\zeta_i\}$ is developed as follows. Since $E[U] < 0$ and $P(U > 0) > 0$, there exists an $\alpha_0 > 0$ such that $E[e^{\alpha_0 U}] = 1$. Define the process $\xi_i^* = e^{\alpha_0 \xi_i}$, $i \ge 0$, with the property

$$E[\xi_{i+1}^* \mid \xi_i^*] = E[e^{\alpha_o U_i} \xi_i^* \mid \xi_i^*] = \xi_i^* E[e^{\alpha_0 U}] = \xi_i^* \ .$$

Together with our assumptions on U, this shows that $\{\xi_i^*\}$ is a uniformly integrable martingale, so we have

$$\begin{aligned} P\left(\sup_{i\ge 0} \xi_i \ge x\right) &= P\left(\sup_{i\ge 0} \xi_i^* \ge e^{\alpha_0 x}\right) \\ &\le e^{-\alpha_0 x} E[\xi_0^*] = E[e^{-\alpha_0 (x - \xi_0)}] \ , \end{aligned} \tag{3.5}$$

where the inequality follows from Doob's martingale inequality (see, for example, section 35 in Billingsley (1986)).

We now use (3.5) to get similar bounds for the *busy periods* of $\{\zeta_i\}$. In analogy with queueing applications, we say that steps i_1 through i_2, $i_2 > i_1$, comprise a busy period if $\{\zeta_i\}$ moves away from the origin at step $i_1 \ge 1$ and makes its first subsequent return to the origin at step i_2, i.e., $\zeta_{i_1 - 1} = 0$, $\zeta_j > 0$, $i_1 \le j < i_2$, and $\zeta_{i_2} = 0$. The process is *idle* while it resides at the origin. We want a probability bound on the

maximum value of the process during a busy period B. For this purpose, we make use of the fact that, away from the origin, $\{\zeta_i\}$ behaves as an unrestricted random walk. In particular, the conditional probability that, given the first jump $U_{i_1} > 0$, $\{\zeta_i\}$ exceeds level x before its next return to the origin is the same as the probability that, starting in state U_{i_1}, the unrestricted version $\{\xi_i\}$ exceeds level x before its first passage to a point at or below the origin. As an easy consequence of (3.3) and (3.5), we have that, for a randomly chosen busy period B of $\{\zeta_i\}$,

$$P\left(\sup_{i\in B}\zeta_i > x\right) \le E[e^{-\alpha_0(x-U)}|U>0] \le e^{-\alpha_0 x}E[e^{\alpha_0 X}|X>K] = O(e^{-\alpha_0 x}), \tag{3.6}$$

since $X \ge U \ge X - K$ and X is a binomial random variable with a mean bounded independently of N.

Our primary interest is in the behavior of $\{\zeta_i\}$ over a finite (and large) number of steps. It is convenient to let N denote the number of steps, since in later applications of the results below, N will also denote the ring size. For example, a bound on $P(\sup_{1\le i\le N}\zeta_i > \alpha \ln N)$, $\alpha > 0$, will be useful. To get such a bound, note that there are at most $N/2$ busy periods in the first N steps of $\{\zeta_i\}$. Then, by (3.6),

$$P\left(\sup_{1\le i\le N}\zeta_i > x\right) \le \frac{N}{2}P\left(\sup_{i\in B}\zeta_i > x\right) = O(e^{-\alpha_0 x+\ln N}) \ .$$

Thus, for any $\gamma > 0$, we can choose $x = x(N) = \alpha \ln N$ with $\alpha = \alpha(\gamma)$ sufficiently large that

$$P\left(\sup_{1\le i\le N}\zeta_i > \alpha\ln N\right) = O(e^{-\gamma\ln N}) = O(N^{-\gamma}) \ . \tag{3.7}$$

Consider next the duration D of busy period B.

LEMMA 3.3. *There exists an $\eta_0 > 0$ such that*

$$P(D > y) = O(e^{-\eta_0 y}) \ .$$

Proof. Let $\{U_i\}$ be the common sequence generating both $\{\zeta_i\}$ and $\{\xi_i\}$, $\zeta_0 = \xi_0 = 0$, and suppose the first busy period B_1 of $\{\zeta_i\}$ begins at step $\ell \ge 1$. Let D_1 be the duration of B_1. It is easy to check that, for any integer $y \ge 1$, the event $\{\zeta_i > 0 \text{ for all } i, \ell \le i \le \ell + y\}$ implies the event $\{\xi_{\ell+y} \ge \xi_\ell\}$. Busy periods are i.i.d. and $P(\xi_{\ell+y} \ge \xi_\ell)$ does not depend on ℓ, so

$$\begin{aligned} P(D > y) = P(D_1 > y) &\le P(\xi_{\ell+y} \ge \xi_\ell) \\ &\le P(\xi_y \ge 0). \end{aligned} \tag{3.8}$$

By Lemma 3.1, we obtain that, for any $\epsilon > 0$, there exists an $\alpha > 0$ such that

$$P(\xi_y > (1-\epsilon)E[\xi_y]) = O(e^{\alpha E[\xi_y]}) \ , \tag{3.9}$$

with $E[\xi_y] = yE[U] < 0$. To see this, we need only observe that the U_i and hence ξ_y can be expressed as sums of independent 0-1 random variables. Put $\epsilon = 1$ in (3.9) and conclude that, for some $\alpha_1 > 0$,

$$P(\xi_y \ge 0) = O(e^{\alpha_1 yE[U]}) \ . \tag{3.10}$$

Together with (3.8), this proves the lemma. □

4. Admission policy. The proof of Theorem 1.1 will use the admission policy of this section. Before presenting the policy, however, we will briefly review how it is applied in the general argument.

The proof of Theorem 1.1 estimates the expected value of the sum $S \equiv S(N,T)$ of waiting times under the greedy policy in an interval of length $T = \Theta(N^3)$, assuming that the state of the queues at the beginning of the interval is a sample from the stationary distribution. For convenience, we take $[0,T]$ as the interval. To make use of the estimate, observe that in the stationary regime, $E[Q_i(t)] = E[Q]$, so by (2.2) $E[S] = NTE[Q]$ and

$$E[Q] = \frac{E[S]}{NT} \ . \tag{4.1}$$

We will prove that, under the admission policy defined below, the sum $\tilde{S}$ of waiting times over $[0,T]$ satisfies $E[\tilde{S}] = O(N^3)$. By Theorem 2.1, $E[S] \le E[\tilde{S}]$, so substitution into (4.1) proves $E[Q] = O(1/N)$, since $T = \Theta(N^3)$. Then Theorem 1.1 is proved.

We now discuss the admission policy, Algorithm A, shown in Fig. 3, previewing as we go along the properties of the algorithm that must be proved in the probabilistic analysis of the next section. The algorithm is based on various constants and structures determined by λ and μ, which we describe first. The algorithm takes as input an $\epsilon > 0$ such that $\mu(1-2\epsilon) > \lambda$, and reserves a sequence of $2\epsilon N$ cells of the ring as nearly equally spaced as possible. (The lengths of adjacent intervals between reserved cells differ by at most 1.) Call the odd-numbered cells of this sequence *initialization* (I) cells, and the even-numbered cells *clean-up* (CU) cells. *Regular* cells are those that are neither I nor CU cells.

To avoid trivialities and to simplify notation, we assume in what follows that $(2\epsilon)^{-1}$ and ϵN are integers. The reserved (I and CU) cells partition the $(1-2\epsilon)N$ regular cells into $2\epsilon N$ groups C_j, with $(2\epsilon)^{-1} - 1$ cells per group. We also define a partition of the queues into $2\epsilon N$ groups G_j, $1 \le j \le 2\epsilon N$, with $(2\epsilon)^{-1}$ per group. The index j is taken mod $2\epsilon N$ if $j > 2\epsilon N$.

Algorithm A determines the schedule over the time interval $[0, N + bN^3]$, which is partitioned into an initial block B_0 of N steps followed by N^2 blocks $B_1, \ldots, B_{N^2}$ of bN steps each. The parameter b must be chosen sufficiently small; the probabilistic analysis of section 5 will give an upper bound in terms of μ and λ.

The algorithm is preceded by the following process: independently, at every cell and time step a mark ($\times$) is placed with probability μ/N; these marks are superposed on the input arrival pattern. If, by Algorithm A, a packet φ is placed in cell j at time t, then the first $\times$ after time t in column j signals the departure of φ from the ring (these particular $\times$'s correspond to bullets in Fig. 2). By the memoryless property of the geometric distribution of the times between successive $\times$'s in any column, this rule for determining departures yields geometric transit times on the ring, as desired.

With this set-up, the algorithm is as follows (see Fig. 3). First, the interval $[0,N]$ of B_0 is devoted solely to the accumulation of empty CU cells, to be used as described later, starting at time N. No admissions to the ring are scheduled during $[0,N]$. This is for convenience only; our asymptotic results would not change if such scheduling were allowed.

At time N, the algorithm partitions the empty CU cells into sequences σ_k, $1 \le k \le a \ln N$, as nearly equal in length as possible (see step 1 in Fig. 3), for a constant a sufficiently large to be determined by the probabilistic analysis. Apart from their

size and number, the sequences σ_k can be chosen arbitrarily from among the empty CU cells.

The remainder of step 1 assigns I cells starting at time N to just those packets in the initial state plus those that arrived in $[0, N]$. The jth I cell admits the packets in the ϵ^{-1} queues of $G_{2j-1} \cup G_{2j}$ at time N; it serves these queues in a round-robin sequence; i.e., a $(k+1)$st packet from one of the queues is not admitted until at least k visits have been made to the other queues (admitting a packet at each visit if one is there).

Note that the I cells work in parallel with the other cells that serve the arrivals in $B_1, \ldots, B_{N^2}$. The probabilistic analysis will use elementary bounds to show that the expected total waiting time of packets served by I cells is negligible, i.e., $o(N^3)$.

Almost all of the arriving packets in the B_i, $1 \le i \le N^2$, are assigned by the iterations of step 2 to regular cells during the interval $[N, N + bN^3]$ (see Fig. 3). Arrivals in B_i are assigned to cells at time $N + (i-1)bN$, $1 \le i \le N^2$. At this time, a regular cell is called *available* if its column segment in B_{i-1} has at least one $\times$, its column segments in B_{i-2} and B_{i-3} have no $\times$, and the cell has not already been assigned to an arrival in B_i (if $i = 2$, then the reference to B_{i-3} is omitted, and if $i = 1$, the references to both B_{i-2} and B_{i-3} are omitted). Examples are given in Fig. 4, which are referenced again at the end of this section. Step 2 scans the groups G_j in left-to-right order beginning with G_1. Assume for simplicity that the cell group C_j is lined up in front of the queues in G_j so that the last cell of C_j is in front of the last queue in G_j (recall that C_j has one fewer cell than G_j has queues). This is the alignment assumed in step 2 of Fig. 3.

For $j = 1, \ldots, 2\epsilon N$, the arrivals as yet unassigned to $G_1, \ldots, G_j$ are assigned in any order to the available cells of C_{j+1} until either the former or latter set is empty, whichever occurs first. At the end of this process, there may still be unassigned arrivals in B_i; these are called *leftover* packets. Also, there may have been instances where an arrival was assigned to a cell more than bN cells (time units) away. These assignments are discarded and the corresponding packets are left unassigned throughout $[0, N + bN^3]$. The restriction to cells that are both available (in the above limited sense) and not too far from the arrivals assigned to them guarantees that Algorithm A makes valid assignments. We will verify this fact after we describe the remainder of the algorithm.

The probabilistic analysis will show that, for each block B_i, the number of available cells in the C_j's is sufficiently large to ensure an $O(N)$ expected total waiting time for the arrivals assigned in step 2. Then, for all N^2 blocks, the total waiting time is $O(N^3)$, as desired. The analysis will then show that the assignment of an arrival to a cell more than bN columns away is so rare that its effect on total waiting time is negligible.

Finally, step 3 of the algorithm takes care of leftover packets by assigning them to the cells of the sequences σ_k. These assignments are organized so that, for each $k = 1, \ldots, a \ln N$, the leftover packets of B_k, $B_{k+a\ln N}$, $B_{k+2a\ln N}$, ... are all assigned to cells in the same sequence σ_k. Thus, for $r \ge 0$, the leftover packets in $B_{k+ra\ln N}, \ldots, B_{k+(r+1)a\ln N-1}$ are served in parallel by disjoint regions of the ring.

The probabilistic analysis will show that, except for a negligible fraction of the leftover packets, all of those admitted by the cells of σ_k from $B_{k+ra\ln N}$ for any $r \ge 0$ will have departed when it is time to start admitting the arrivals in $B_{k+(r+1)a\ln N}$ into the cells of σ_k. In addition, the analysis will show that the expected total wait of the leftover packets in B_i is $O(N)$, and hence the expected total wait for leftovers from all N^2 blocks is $O(N^3)$, as desired.

Algorithm A

Input: N, a, b, ϵ, an initial state and sets of arrivals and marks ($\times$'s) over $[0, N + bN^3]$

1. (i) At time N, the empty CU cells are partitioned into sequences σ_k, $1 \le k \le a \ln N$, whose lengths differ by at most 1.
   (ii) For $j = 1, \ldots, \epsilon N$, the jth I cell admits just those packets appearing in the ϵ^{-1} queues of $G_{2j-1} \cup G_{2j}$ at time N. For each j, these queues are served by a round-robin starting at time N.

For $i = 1, \ldots, N^2$, the following two steps are performed.

2. (i) Assume that the queues in G_j are aligned with the cells of C_j, at the time B_i begins. For $j = 1, \ldots, 2\epsilon N$, the as yet unassigned arrivals in the queues of $G_1, \ldots, G_j$ are assigned to the available cells in C_{j+1} until the former or the latter are exhausted, whichever occurs first ($C_{2\epsilon N+1} \equiv C_1$).
   (ii) Assignments just made that match an arrival to a cell more than bN columns (cells) away are removed.
3. Let integer k satisfy $1 \le k \le a$ and $i = ma + k$ for some integer $m \ge 0$. Then the leftover packets, if any, of B_i are admitted according to the greedy rule by the empty CU cells of σ_k; admissions stop when there are no more leftovers to admit or when the empty cells of σ_k have been exhausted, whichever occurs first.

FIG. 3. *An admission algorithm.*

It is easy to see that, if Algorithm A always makes valid assignments (loads packets into empty cells), then it is indeed a valid admission policy; the (future) arrivals in B_i are known when assignments are made at the beginning of B_i, but knowledge of future departure times is not used at any point. (This is obvious for steps 1 and 3; it is clear for step 2 as well, since cell availability at the beginning of B_i depends only on departures times in $B_1 \cup \cdots \cup B_{i-1}$.)

It remains to verify that, under Algorithm A, whenever an arrival reaches the cell to which it is assigned by the algorithm, the cell is empty. But suppose that cell j is the available cell assigned by step 2(i) to arrival φ in B_i and that it remains assigned to cell j after step 2(ii). (See Fig. 4 for examples.) Then the earliest that cell j can again become available occurs when assigning arrivals in B_{i+3}; no arrival of B_{i+1} or B_{i+2} can be assigned to cell j by the definition of cell availability and the fact that B_{i-1} has a $\times$ in column j. If cell j is indeed available during the scan of B_{i+3}, then there must be a $\times$ in B_{i+2}. This $\times$ must come after the admission of φ to cell j; otherwise the motion line of φ would span more than bN columns, and this would contradict step 2(ii), where such assignments are removed. Thus, this $\times$ in B_{i+2} guarantees that any packet already in cell j will have departed before cell j is re-used for an arrival in B_{i+3} or some later block.

5. Proof of Theorem 1.1. Recall the general approach outlined at the beginning of the previous section: we prove that $E[\tilde{S}] = O(N^3)$, where $\tilde{S}$ is the sum of the waiting times in $[0, N + bN^3]$ under Algorithm A. It is enough to show, as is done below, that the $O(N^3)$ bound holds for the packets considered by each of the three steps individually.

In what follows, when we say that an event occurs *with high probability*, we mean that it occurs with probability $1 - O(N^{-\gamma})$, where γ can be made as large as desired by a suitable choice of (usually hidden) constants. For example, by the geometric law for transit times V, we have

$$P(V \le dN \log N) = 1 - (1 - \mu/N)^{dN \log N} ,$$

and so $P(V \le dN \log N) \sim 1 - N^{-\mu d}$ as $N \to \infty$. Thus, we can say that transit times are $O(N \log N)$ with high probability; $\gamma = \mu d$ can be made as large as desired by increasing d. Note that m high-probability events occur *jointly* with high probability if m is at most some polynomial in N.

Step 1. Consider the ith queue length $Q_i(N)$ at time N and recall that $Q_i(0)$ has the stationary distribution. Algorithm A admits no packets to the ring in $[0, N]$, so $Q_i(N)$ is $Q_i(0)$ plus the number of arrivals in $[0, N]$ at queue i. At time N, $Q_i^*(N)$ counts the packets waiting at time N in queue i plus the packet, if any, in the I cell that serves queue i in step 1. We have $Q_i^*(N) \le Q_i(N) + 1$, so by Lemma 2.1 and the geometric law of interarrival times, there exists an $\eta > 0$ such that $P(Q_i^*(N) \ge k+1) \le P(Q_i(N) \ge k) = O(e^{-\eta k})$ independently for all queues. Let $\tilde{S}_i^{(1)}$ be the total waiting time of the packets counted by $Q_i(N)$, and let $\mathcal{C}$ be the joint event in which (i) for some $c > 0$ the first $Q_i(N) \wedge c \ln N$ packets waiting in queue i at time N have $O(N \log N)$ transit times, and (ii) at time N the packet, if any, in the I cell serving queue i has $O(N \log N)$ remaining transit time. By the geometric law for transit times, $\mathcal{C}$ has high probability.

Since each I cell serves a constant number of queues in a round-robin sequence, the kth packet in any queue must wait $k \cdot O(N \log N)$ time if $\mathcal{C}$ holds and $k \le c \ln N$. Since no packet can wait more than $bN^3 + N$ time, we obtain the bound

$$\begin{aligned} E[\tilde{S}_i^{(1)} | \mathcal{C}] &= \sum_{1 \le k \le c \ln N} k \cdot O(N \log N) \quad + \quad O(N^3) \cdot E(Q_i(N) - c \ln N)^+ \\ &= O(N \log^3 N) + O(N^{-\eta c}) \cdot O(N^3) \\ &= O(N \log^3 N) \end{aligned}$$

by choosing c large enough. If $\mathcal{C}$ does not hold, we use the $O(N^3)$ trivial bound for all packets to obtain $E[\tilde{S}_i^{(1)} | \overline{\mathcal{C}}] = O(N^3)$, since $E[Q_i(N)] = O(1)$. But $\overline{\mathcal{C}}$ has low probability, so $E[\tilde{S}_i^{(1)}] = O(N \log^3 N)$. Since $\mathcal{C}$ holds simultaneously for all i with high probability, we can conclude that

$$E[\tilde{S}^{(1)}] = N E[\tilde{S}_i^{(1)}] = O(N^2 \log^3 N) = O(N^3),$$

as desired.

Step 2. We analyze the left-to-right scan of the sets G_j in B_i and C_j in B_{i-1}, and bound first the expected total waiting time of the packets that are assigned in step 2(i). Define the Lindley process

$$\zeta_0 = 0, \quad \zeta_j = (\zeta_{j-1} + U_j)^+, \quad j = 1, \ldots, \epsilon N,$$

where $U_j = X_j - Y_j$, X_j is the number of arrivals in G_j and Y_j is the number of available columns in C_{j+1} at the start of the jth iteration in step 2. It is easy to see that, among the arrivals already scanned in $G_1, \ldots, G_j$, ζ_j gives the number as yet unassigned at the start of the $(j+1)$st iteration. Thus, $(2\epsilon)^{-1}(\zeta_1 + \cdots + \zeta_{2\epsilon N})$ is the cumulative waiting time $\tilde{S}_i^{(2)}$ of the arrivals assigned in B_i, not counting the times spent waiting by these arrivals in their initial and final blocks. The latter times are bounded by $2(2\epsilon)^{-1} = \epsilon^{-1}$, so

$$\tilde{S}_i^{(2)} \le (2\epsilon)^{-1}(\zeta_1 + \cdots + \zeta_{2\epsilon N}) + \epsilon^{-1}(X_1 + \cdots + X_{2\epsilon N}).$$

But $E[X_j] = \lambda b (2\epsilon)^{-1}$, and if $E[U_j] < 0$ then $E[\zeta_j] = O(1)$, by Lemma 3.2, so that $E[\tilde{S}_i^{(2)}] = O(N)$ and hence the expected total wait of assigned packets summed over all i is $O(N^3)$, as desired. Thus, it remains to prove that $E[U_j] < 0$.

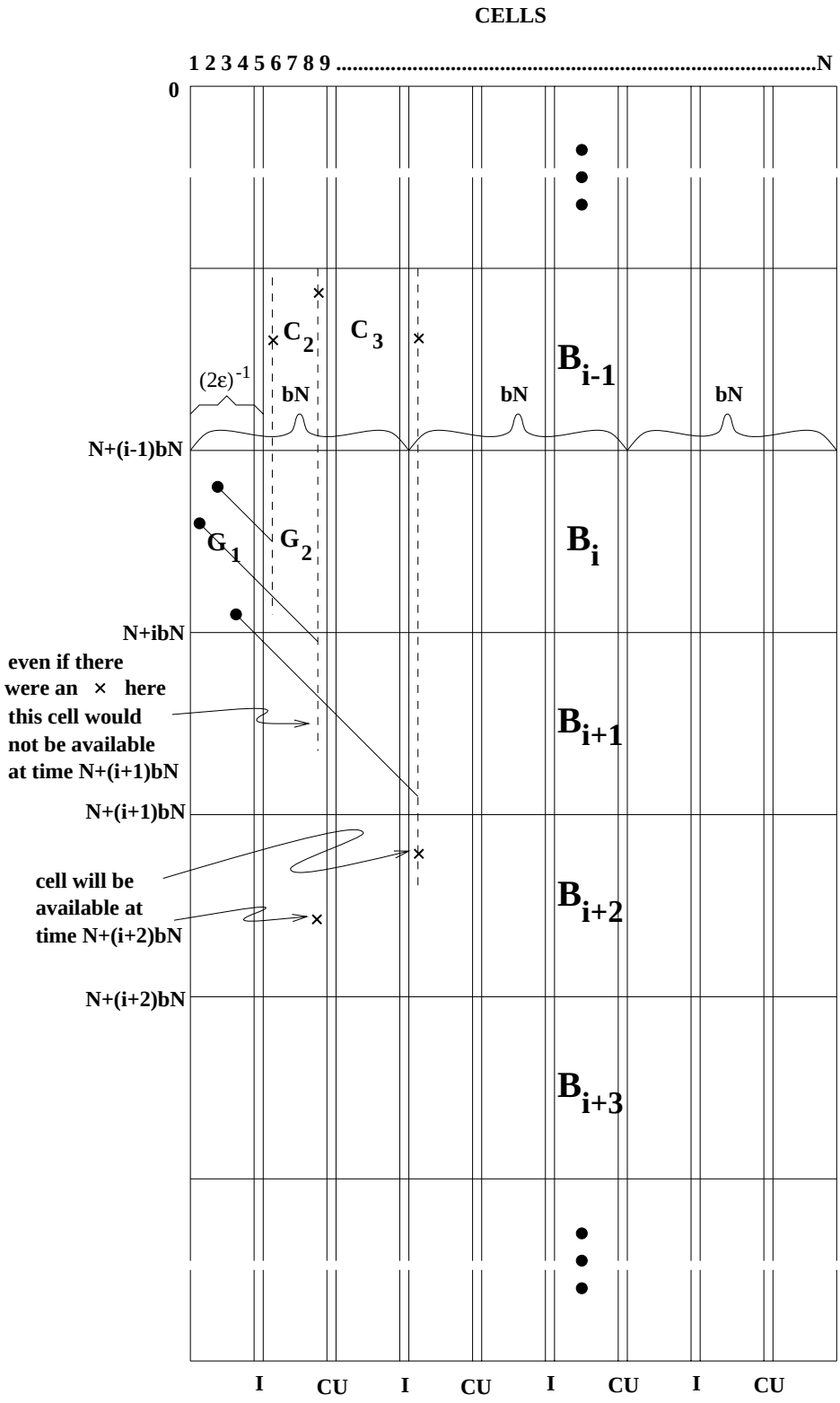


FIG. 4. *Admission algorithm.*

We need to verify that $P(A_k) > \lambda b/(1-2\epsilon)$, where A_k is the event in which column k is available when assignments to arrivals in B_i begin; for then, since there are $(2\epsilon)^{-1}-1$ columns in the C_j,

$$E[Y_j] > \frac{\lambda b}{1-2\epsilon}[(2\epsilon)^{-1}-1] = (2\epsilon)^{-1}\lambda b = E[X_j],$$

and hence $E[U_j] < 0$. We consider B_i for $i \geq 4$; the cases $i = 1, 2, 3$ are similar. We need only observe that A_k holds if there is at least one $\times$ in a column of B_{i-1} and none in B_{i-2} and B_{i-3}, so

$$P(A_k) = [1-(1-\mu/N)^{bN}](1-\mu/N)^{2bN} \sim (1-e^{-\mu b})e^{-2\mu b}$$

as $N \to \infty$. Then for all N sufficiently large,

$$b < \frac{1}{5}\frac{\mu(1-2\epsilon)-\lambda}{(1-2\epsilon)\mu^2}$$

is enough to ensure that $P(A_k) > \lambda b/(1-2\epsilon)$.

It remains to estimate the added total waiting time of the packets that were assigned but then unassigned in step 2. But by Lemma 3.3 the probability that a packet is assigned to a $\times$ at least bN columns (and hence $\Omega(N)$ groups C_j) away is exponentially small in N. It follows that the expected added total waiting time of such packets is $o(1)$, since the number and maximum wait of such packets are both bounded by polynomials in N. Thus, the expected total wait of all packets examined in step 2 is $E[\tilde{S}^{(2)}] = O(N^3)$, as desired.

Step 3. Let $k_i = (i-1) \bmod (a \ln N) + 1$, and note that, by step 3, the leftover packets of B_i should go into the cells of σ_{k_i}.

We argue first that, by an application of Lemma 3.1, at time N there are at least $\frac{\epsilon}{2}(1-e^{-\mu})N$ empty CU cells with very high probability (i.e., with probability $1-O(e^{-\Omega(N)})$); thus, with very high probability, any existing leftover packets are assigned to the $a \log N$ sequences σ_k, which have $\Omega(N^\beta)$ cells each for every β, $0 < \beta < 1$.

For definiteness, choose $\beta = 1/2$ and let $\mathcal{E}_i$ be the event in which the leftover packets of B_i number fewer than $N^{1/2}$ and each has a transit time at most $bN \cdot a \ln N - N$. In this event, the waiting time of each leftover packet is at most N and the leftover packets of B_i leave the CU cells of σ_{k_i} empty by the time the next set of leftover packets (those in $B_{i+a\ln N}$) have to be scheduled in the cells of σ_{k_i}. By (3.7) and the geometric law for transit times, $\mathcal{E}_i$ holds with high probability for all a large enough. There are only N^2 such events, so the combined event $\mathcal{E} = \bigcap_{i=0}^{N^2} \mathcal{E}_i$ also holds with high probability, where $\mathcal{E}_0$ is the event in which there exist at least $\frac{\epsilon}{2}(1-e^{-\mu})N$ empty CU cells at time N.

Now suppose $\mathcal{E}$ holds. Then, since a leftover packet waits at most N and there are N^2 blocks, the conditional expected total waiting time is $O(N^3)$ times the conditional expected number of leftover packets per block B_i, i.e., $O(N^3) \cdot E[\zeta_N|\mathcal{E}]$. But

$$E[\zeta_N|\mathcal{E}] = E[\zeta_N|\zeta_N \leq N^{1/2}] \leq E[\zeta_N] = O(1),$$

by Lemma 3.2, so the expected total waiting time of leftover packets is $O(N^3)$ when $\mathcal{E}$ holds.

Given that $\mathcal{E}$ does not hold, we use the trivial polynomial bounds $O(N^4)$ and $O(N^3)$ on the total number of leftover packets and the waiting time of each. Since $\mathcal{E}$ fails with low probability, a can be chosen large enough so that $P(\mathcal{E}) = 1 - O(N^{-4})$. Thus, the expected total waiting time of leftover packets is

$$O(N^3) + (1 - P(\mathcal{E}))O(N^7) = O(N^3)$$

and the theorem is proved. □

6. Final remarks. A close look at the analysis in section 5 shows that it is possible to prove a stronger version of Theorem 1.1 in which the dependence of the hidden multiplicative constant on λ and μ is specified: there exists a universal constant α such that, for N sufficiently large,

$$E[W] \leq \frac{\alpha}{(1-\rho)^2} \ . \tag{6.1}$$

The details of a proof of this result have been omitted because no new ideas are needed, and because the added clutter makes the proof significantly harder to follow. In broad outline, a proof can begin with the observation that if (6.1) can be proved for the expected waiting time $E[W^{(2)}]$ of packets assigned in step 2, then changing only the constant α, it must also hold for $E[W]$. This is not difficult to verify using the probability bounds of sections 2 and 3, and the arguments in section 5.

It is then not difficult to verify that, within a constant factor independent of λ and μ, $E[W^{(2)}]$ is the expected waiting time in a G/G/1 queue with arrivals in each time slot having a binomial distribution with mean λb and service times having a geometric distribution with rate parameter $\mu' \approx (1-2\epsilon)(1-e^{-\mu b})e^{-2\mu b}$. For large N, the queue is asymptotically an M/G/1 queue, so by classical results, we get (Kleinrock (1975), section 5.7)

$$E[W^{(2)}] = O\left(\frac{1}{(1-\rho')\mu'}\right),$$

where $\rho' = \lambda'/\mu'$ and $\lambda' = \lambda b$.

As in the analysis of Step 2 in section 5, we again choose $b = \Theta(\frac{\mu-\lambda}{\mu^2})$, and so $1 - \rho' = \Omega(1-\rho)$ and $\mu' = \Omega(\mu b) = \Omega(1-\rho)$. Thus, $E[W^{(2)}]$ and hence $E[W]$ has a bound of the form (6.1).

Asymptotics in N pose intriguing open problems for transit-time distributions other than the geometric. The uniform distribution on $\{1, \ldots, N-1\}$ is of particular interest; extensive simulations by Coffman et al. (1995) give convincing evidence that the bounds in Theorem 1.1 hold for this case as well, but no proof has yet been found.

Finally, in keeping with our Markov arrival and transit-time assumptions, it would be interesting to study asymptotic behavior in the generalization of rings to toroidal arrays of processors (see Leighton (1990, 1992)). Much is known about regular (open) arrays, as can be seen from the recent work of Mitzenmacher (1994) and Kahale and Leighton (1995), who give references to the earlier work on this problem. But the analysis of toroidal arrays seems to require different methods.

Acknowledgment. We are grateful to M. Harchol-Balter for improvements to an early version of this paper, and to I. Telatar and A. Weiss for helpful discussions. The second author also thanks DIMACS for its hospitality.

REFERENCES

N. Alon and J. H. Spencer (1991), *The Probabilistic Method*, John Wiley, New York.
S. Asmussen (1987), *Applied Probability and Queues*, John Wiley, New York.
L. A. Barroso and M. Dubois (1992), *The performance of Cache-Coherent ring-based multiprocessors*, in Proc. 20th Ann. Internat. ACM Symp. Comp. Arch., ACM, New York, pp. 268–277.
P. Billingsley (1986), *Probability and Measure*, 2nd ed., John Wiley, New York.
E. G. Coffman, Jr., E. N. Gilbert, A. G. Greenberg, F. T. Leighton, P. Robert, and A. L. Stolyar (1995), *Queues served by a rotating ring*, Stochastic Models, 11, pp. 371–394.
L. Georgiadis, W. Szpankowski, and L. Tassiulas (1997), *Stability analysis of quota allocation access protocols in ring networks with spatial reuse*, IEEE Trans. Inform. Theory, 43, pp. 923–937.
N. Kahale and F. T. Leighton (1995), *Greedy dynamic routing on arrays*, in Proc. 6th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, pp. 558–566.
L. Kleinrock (1975), *Queueing Systems, Vol.* I, John Wiley, New York.
F. T. Leighton (1990), *Average case analysis of greedy routing algorithms on arrays*, in Proc. 2nd Ann. ACM Symp. Parallel Algs. Arch., ACM, New York, pp. 2–10.
F. T. Leighton (1992), *Introduction To Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann, San Mateo, CA.
M. Mitzenmacher (1994), *Bounds on the greedy algorithm for array networks*, in Proc. 6th Ann. ACM Symp. Parallel Alg. Arch., ACM, New York, pp. 346–353.
N. U. Prabhu (1965), *Stochastic Processes*, Macmillan, New York.
B. Van Arem and E. A. Van Doorn (1990), *Analysis of a queueing model for slotted ring networks*, Comput. Networks ISDN Systems, 20, pp. 309–314.

SIAM J. COMPUT.
Vol. 27, No. 5, pp. 1237–1261, October 1998
© 1998 Society for Industrial and Applied Mathematics
002

APPROXIMATION ALGORITHMS FOR PSPACE-HARD HIERARCHICALLY AND PERIODICALLY SPECIFIED PROBLEMS*

MADHAV V. MARATHE†, HARRY B. HUNT III‡, RICHARD E. STEARNS‡, AND VENKATESH RADHAKRISHNAN§

Abstract. We study the efficient approximability of basic graph and logic problems in the literature when instances are specified hierarchically as in [T. Lengauer, *J. Assoc. Comput. Mach.*, 36(1989), pp. 474–509] or are specified by one-dimensional finite narrow periodic specifications as in [E. Wanke, *Paths and cycles in finite periodic graphs*, in Lecture Notes in Comp. Sci. 711, Springer-Verlag, New York, 1993, pp. 751–760]. We show that, for most of the problems Π considered when specified using *k-level-restricted* hierarchical specifications or k-narrow periodic specifications, the following hold.

(i) Let ρ be any performance guarantee of a polynomial time approximation algorithm for Π, when instances are specified using standard specifications. Then $\forall \epsilon > 0$, Π has a polynomial time approximation algorithm with performance guarantee $(1 + \epsilon)\rho$.

(ii) Π has a polynomial time approximation scheme when restricted to planar instances.

These are the first polynomial time approximation schemes for PSPACE-hard hierarchically or periodically specified problems. Since several of the problems considered are PSPACE-hard, our results provide the first examples of natural PSPACE-hard optimization problems that have polynomial time approximation schemes. This answers an open question in Condon et al. [*Chicago J. Theoret. Comput. Sci.*, 1995, Article 4].

Key words. hierarchical specifications, periodic specifications, PSPACE-hardness, approximation algorithms, computational complexity, CAD systems, VLSI design

AMS subject classifications. 68R10, 68Q15, 68Q25, 05C40

PII. S0097539795285254

1. Introduction and motivation.

Many practical applications of graph theory and combinatorial optimization in CAD systems, mechanical engineering, VLSI design, and software engineering involve processing large objects constructed in a systematic manner from smaller and more manageable components [13]. An important example of this occurs in VLSI technology. Currently, VLSI circuits can consist of millions of transistors. But such large circuits usually have a highly regular design and consequently are defined systematically, in terms of smaller circuits. As a result, the graphs that abstract the structure and operation of the underlying circuits (designs) also have a regular structure and are defined systematically in terms of smaller graphs. Methods for describing large but regular objects by small descriptions are referred to as *succinct specifications*. Over the last 20 years several theoretical models have been put forward to succinctly represent objects such as graphs and circuits.

*Received by the editors April 20, 1995; accepted for publication (in revised form) June 21, 1996; published electronically May 19, 1998. A preliminary version of this paper appeared in Proc. 26th ACM Ann. Symp. on Theory of Computing, 1994, pp. 468–477.

http://www.siam.org/journals/sicomp/27-5/28525.html

†Los Alamos National Laboratory, P.O. Box 1663, MS B265, Los Alamos, NM 87545 (marathe@lanl.gov). Part of this research was done when the author was at SUNY-Albany and was supported by NSF grant CCR 94-06611. This work was supported by Department of Energy contract W-7405-ENG-36.

‡Department of Computer Science, State University of New York (SUNY) at Albany, Albany, NY 12222 (hunt@cs.albany.edu, res@cs.albany.edu). This research was supported by NSF grants CCR 89-03319 and CCR 94-06611.

§Mailstop 47LA-2, Hewlett-Packard Company, 19447 Pruneridge Avenue, Cupertino, CA 95014-9913 (rven@cup.hp.com). Part of the research was done when the author was at SUNY-Albany, and was supported by NSF grant CCR 89-03319.

(See, for example, [5, 6, 11, 19, 23, 26, 27, 34, 48, 50, 52, 57].) Here, we study two kinds of succinct specifications, namely, hierarchical and periodic specifications.

Hierarchical specifications allow the overall design of an object to be partitioned into the design of a collection of modules, which is a much more manageable task than producing a complete design in one step. Such a top down (or hierarchical design) approach also facilitates the development of CAD systems, since low-level objects can be incorporated into libraries and can thus be made available as submodules to designers of large scale objects. Other areas where hierarchical specifications have found applications are VLSI design and layout [18, 19, 47, 55], finite element analysis, software engineering [13], and datalog queries (see [18, 29, 43] and the references therein). Periodic specifications can also be used to define large scale systems with highly regular structures. Using periodic specifications, large objects are described as repetitive connections of a basic module. Frequently, the modules are connected in a linear fashion, but the basic modules can also be repeated in two- or higher dimensional patterns. Periodic specifications are also used to model time variant problems, where the constraints or demands for any one period are the same as those for preceding or succeeding periods. Periodic specifications have applications in such diverse areas as operations/management [46], transportation planning [18, 43, 48], parallel programming [18, 26], and VLSI design [23, 24].

Typically, the kinds of hierarchical and periodic specifications studied in the literature are generalizations of standard specifications used to describe objects. An important feature of both these kinds of specifications is that they can be much more concise in describing objects than standard specifications. In particular, the size of an object can be exponential in the size of its periodic or hierarchical specifications. As a result of this, problems for hierarchically and periodically specified inputs often become PSPACE-hard, NEXPTIME-hard, etc.

In this paper, we concentrate our attention on

1. the hierarchical specifications of Lengauer [31, 34, 35] (referred to as L-specifications) and
2. the one-dimensional finite periodic specifications of Gale and Wanke [10, 58] (referred to as 1-FPN-specifications).

Both of these specifications have been used to model problems in areas such as CAD systems and VLSI design [35, 32, 36], transportation planning [10], parallel programming [58], etc. We give formal definitions of these specifications in sections 4 and 5.

Let Π be a problem posed for instances specified using *standard* specifications. For example, if Π is a satisfiability problem for CNF formulas, the standard specification is sets of clauses, with each clause being a set of literals. Similarly, if Π is a graph problem, the adjacency matrix representation or the adjacency list representation of the edges in the graph are standard specifications. For the rest of the paper, we use

1. L-Π to denote the problem Π, when instances are specified using the hierarchical specifications of Lengauer [35] (see Definition 4.1), and
2. 1-FPN-Π to denote the problem Π, when instances are specified using the one-dimensional finite periodic specifications of Wanke [58] (see Definition 5.1).

Thus for example, L-3SAT denotes the problem 3SAT when instances are specified using L-specifications, and 1-FPN-3SAT denotes the problem 3SAT when instances are specified using 1-FPN-specifications. For the rest of this paper, we use the term succinct specifications to mean both L-specifications and 1-FPN-specifications.

2. Summary of results. In this paper, we discuss a natural syntactic restriction on the L-specifications and call the resulting specifications *level-restricted* specifications. (For 1-FPN-specifications our notion of level-restricted specifications closely coincides with Orlin's notion of narrow specifications [48].) Most of the problems considered in this paper are PSPACE-hard even for level-restricted specifications (see [37, 44, 48]). Consequently, we focus our attention on devising polynomial time approximation algorithms for level-restricted L- or 1-FPN-specified problems. Recall that an approximation algorithm for a minimization problem[1] Π provides a *performance guarantee* of ρ if for every instance I of Π, the solution value returned by the approximation algorithm is within a factor ρ of the optimal value for I. A *polynomial time approximation scheme* (PTAS) for problem Π is a family of algorithms such that, given an instance I of Π, $\forall \epsilon > 0$, there is a polynomial time algorithm in the family that returns a solution which is within a factor $(1+\epsilon)$ of the optimal value for I. The main contributions of this paper include the following.

(i) We design polynomial time approximation algorithms (for arbitrary instances) and approximation schemes (for planar instances) for a variety of natural PSPACE-hard problems specified using level-restricted L- or 1-FPN-specifications. These are the first PTASs in the literature for "hard" problems specified using either L- or 1-FPN-specifications. To obtain our results we devise a new technique called the *partial expansion*. The technique has two desirable features. First, it works for a large class of problems and second, it works well for *both* L-specified and 1-FPN-specified problems.

(ii) For problems specified using level-restricted L- or 1-FPN-specifications, we devise polynomial time approximation algorithms with performance guarantees that are asymptotically equal to the best possible performance guarantees for the corresponding problems specified using standard specifications.

(iii) The results presented in this paper are a step towards finding sufficient syntactic restrictions on the L- or 1-FPN-specifications that allow us to specify a number of realistic designs in a succinct manner while making them amenable for rapid processing.

Our results provide the first examples of *natural* PSPACE-complete problems whose optimization versions have PTASs. Thus they affirmatively answer the question posed by Condon et al. [8] of whether there exist *natural* classes of PSPACE-hard optimization problems that have PTASs.

2.1. The meaning of approximation algorithms for succinctly specified problems. When objects are represented using L- or 1-FPN-specifications, there are several possible ways of defining what it means to "design a polynomial time approximation algorithm." Corresponding to each decision problem Π, specified using either L- or 1-FPN-specifications, we consider four variants of the corresponding optimization problem. We illustrate this with an example.

Example 1. Consider the minimum vertex cover problem, where the input is an L-specification of a graph G. We provide efficient algorithms for the following versions of the problem.

1. *The construction problem*: Output an L-specification of the set of vertices in the approximate vertex cover $\mathcal{C}$.
2. *The size problem*: Compute the size of the approximate vertex cover $\mathcal{C}$ for G.

[1] A similar definition can be given for maximization problems.

3. *The query problem*: Given any vertex v of G and the path from the root to the node in the *hierarchy tree* (see section 2 for the definition of hierarchy tree) in which v occurs, determine whether v belongs to the vertex cover $\mathcal{C}$.

4. *The output problem*: Output the approximate vertex cover $\mathcal{C}$.

Note that our algorithms for the four variants of the problem apply to the *same* vertex cover $\mathcal{C}$. Our algorithms for (1), (2), and (3) above run in time *polynomial in the size of the L-specification* rather than the size of the graph obtained by expanding the L-specification. Our algorithm for (4) runs in time linear in the size of the expanded graph but uses space which is only polynomial in the size of the L-specification. □

Analogous variants of approximation algorithms can be defined for problems specified using 1-FPN-specifications. Therefore, we omit this discussion.

These variants are natural extensions of the definition of approximation algorithms for problems specified using standard specifications. This can be seen as follows: when instances are specified using standard specifications, the number of vertices is polynomial in the size of the description. Given this, any polynomial time algorithm to determine if a vertex v of G is in the approximate minimum vertex cover can be easily modified to obtain a polynomial time algorithm that lists all the vertices of G in the approximate minimum vertex cover. Thus, in the case when inputs are specified using standard specifications, (3) can be used to solve (2) and (4) in polynomial time. The above discussion also shows that given an optimization problem specified using standard specifications, variants (1), (3), and (4) discussed above are polynomial time interreducible.

The approximation algorithms given in this paper have another desirable feature. For an optimization problem or a query problem, our algorithms use space and time, which is a low level polynomial in the size of the hierarchical or the periodic specification. This implies that for graphs of size N, which are specified using specifications of size $O(\text{polylog}\, N)$, the time and space required to solve problems is only $O(\text{polylog}\, N)$. Moreover, when we need to output the subset of vertices, subset of edges, etc. corresponding to a vertex cover, maximum cut, etc., in the expanded graph, our algorithms take essentially the same time but substantially less (often logarithmically less) space than algorithms that work directly on the expanded graph. The graphs obtained by expanding hierarchical or periodic descriptions are frequently too large to fit into the main memory of a computer [31]. This is another reason for designing algorithms which exploit the regular structure of the underlying graphs. Indeed, most of the standard algorithms in the literature assume that the input completely resides in the main memory. As a result, even the most efficient algorithms incur a large number of page faults while executing on the graphs obtained by expanding the hierarchical or periodic specifications. Hence, algorithms designed for solving problems for graphs or circuits represented in a standard fashion are often impractical for succinctly specified graphs. We refer the reader to [31, 36] for more details on this topic.

The rest of the paper is organized as follows. Section 3 contains discussion of related research. In sections 4, 5, and 6 we give the basic definitions and preliminaries. In section 7 we discuss our approximation algorithms for L-specified problems and 1-FPN-specified problems. Finally, in section 8, we give concluding remarks and directions for future research.

3. Related research. In the past, much work has been done on characterizing the complexity of various problems when instances are specified using L- or 1-FPN-specifications. For periodically specified graphs, several researchers [6, 7, 19, 26, 27, 29, 49, 50] have given efficient algorithms for solving problems such as determining

STRONGLY CONNECTED COMPONENTS, TESTING FOR EXISTENCE OF CYCLES, FINDING MINIMUM COST PATHS BETWEEN A PAIR OF VERTICES, BIPARTITENESS, PLANARITY, and MINIMUM COST SPANNING FORESTS. Orlin [50] and Wanke [58] discuss NP- and PSPACE-hardness results for infinite and finite periodically specified graphs.

For L-specified graphs, Lengauer and Wanke [32, 34, 35] and Williams [59] have given efficient algorithms to solve several graph-theoretic problems including 2-COLORING, MINIMUM SPANNING FORESTS, and PLANARITY TESTING. Lengauer and Wagner [37] show that the following problems are PSPACE-hard when graphs are L-specified: 3-COLORING, HAMILTONIAN CIRCUIT AND PATH, MONOTONE CIRCUIT VALUE PROBLEM, NETWORK FLOW, ALTERNATING GRAPH ACCESSIBILITY, and MAXIMUM INDEPENDENT SET. In [38], Lengauer and Wanke consider a more general hierarchical specification of graphs based on graph grammars and give efficient algorithms for several basic graph-theoretic problems specified using this specification. We refer the reader to [18, 43] for a detailed survey of the work done in the area of hierarchical and periodic specifications.

A substantial amount of research has been done on finding polynomial time approximation algorithms with provable worst case guarantees for NP-hard problems. In contrast, until recently little work has been done towards investigating the existence of polynomial time approximation algorithms for PSPACE-hard problems. As a step in this direction, in [40, 41] we have investigated the existence and nonexistence of polynomial time approximations for several PSPACE-hard problems for L-specified graphs. In [20], we considered geometric intersection graphs defined using the hierarchical specifications (HIL) of Bentley, Ottmann, and Widmayer [5]. There, we devised efficient PTASs for a number of problems for geometric intersection graphs specified using a restricted form of HIL.

Condon et al. [8, 9] also studied the approximability of several PSPACE-hard optimization problems. They characterize PSPACE in terms of probabilistically checkable debate systems and use this characterization to investigate the existence and nonexistence of polynomial time approximation algorithms for a number of basic PSPACE-hard optimization problems.

4. The L-specifications. This section discusses the L-specifications. The following two definitions are essentially from Lengauer [32, 35, 37].

DEFINITION 4.1. *An L-specification* $\Gamma = (G_1, \ldots, G_n)$ *of a graph is a sequence of labeled undirected simple graphs* G_i *called cells. The graph* G_i *has* m_i *edges and* n_i *vertices.* p_i *of the vertices are called pins. The other* $(n_i - p_i)$ *vertices are called inner vertices.* r_i *of the inner vertices are called nonterminals. The* $(n_i - r_i)$ *vertices are called terminals. The remaining* $n_i - p_i - r_i$ *vertices of* G_i *that are neither pins nor nonterminals are called explicit vertices.*

Each pin of G_i *has a unique label, its name. The pins are assumed to be numbered from* 1 *to* p_i*. Each nonterminal in* G_i *has two labels* (v, t)*, a name, and a type. The type* t *of a nonterminal in* G_i *is a symbol from* $G_1, \ldots, G_{i-1}$*. The neighbors of a nonterminal vertex must be terminals. If a nonterminal vertex* v *is of the type* G_j *in* G_i*, then* v *has degree* p_j *and each terminal vertex that is a neighbor of* v *has a distinct label* (v, l) *such that* $1 \leq l \leq p_j$*. We say that the neighbor of* v *labeled* (v, l) matches *the* l*th pin of* G_j*.*

Note that a terminal vertex may be a neighbor of several nonterminal vertices. Given an L-specification Γ, $N = \sum_{1 \leq i \leq n} n_i$ denotes the *vertex number*, and $M = \sum_{1 \leq i \leq n} m_i$ denotes the *edge number* of Γ. The size of Γ, denoted by $size(\Gamma)$, is $N + M$.

DEFINITION 4.2. *Let $\Gamma = (G_1, \ldots, G_n)$ be an L-specification of a graph $E(\Gamma)$ and let $\Gamma_i = (G_1, \ldots, G_i)$. The expanded graph $E(\Gamma)$ (i.e., the graph associated with Γ) is obtained as follows.*

$k = 1$: $E(\Gamma) = G_1$.

$k > 1$: Repeat the following step for each nonterminal v of G_k, say of the type G_j: delete v and the edges incident on v. Insert a copy of $E(\Gamma_j)$ by identifying the lth pin of $E(\Gamma_j)$ with the node in G_k that is labeled (v, l). The inserted copy of $E(\Gamma_j)$ is called a subcell of G_k.

Observe that the expanded graph can have multiple edges although none of the G_i have multiple edges. Here, however, we *only* consider *simple* graphs; i.e., there is at most one edge between a pair of vertices. This means that multi-edges are treated simply as single edges. We assume that Γ is not redundant in the sense that for each j, $1 \leq j \leq n$, there is a nonterminal v of type G_i in the definition of G_j, $j > i$.

The expansion $E(\Gamma)$ is the graph associated with the L-specification Γ with vertex number N. For $1 \leq i \leq n$, $\Gamma_i = (G_1, \ldots, G_i)$ is the L-specification of the graph $E(\Gamma_i)$. Note that the total number of nodes in $E(\Gamma)$ can be $2^{\Omega(N)}$. (For example, a complete binary tree with $2^{\Omega(N)}$ nodes can be specified using an L-specification of size $O(N)$.) To each L-specification $\Gamma = (G_1, \ldots, G_n)$, $n \geq 1$, we associate a labeled rooted unoriented tree $HT(\Gamma)$ depicting the insertions of the copies of the graphs $E(\Gamma_j)$ $(1 \leq j \leq n-1)$, made during the construction of $E(\Gamma)$ as follows (see Figure 4.1).

DEFINITION 4.3. *Let $\Gamma = (G_1, \ldots, G_n)$, $n \geq 1$, be an L-specification of the graph $E(\Gamma)$. The* hierarchy tree *of Γ, denoted by $HT(\Gamma)$, is the labeled rooted unordered tree defined as follows.*

1. *Let r be the root of $HT(\Gamma)$. The label of r is G_n. The children of r in $HT(\Gamma)$ are in* one-to-one *correspondence with the nonterminal vertices of G_n as follows. The label of the child s of r in $HT(\Gamma)$ corresponding to the nonterminal vertex (v, G_j) of G_n is (v, G_j).*

2. *For all other vertices s of $HT(\Gamma)$ and letting the label of $s = (v, G_j)$, the children of s in $HT(\Gamma)$ are in* one-to-one *correspondence with the nonterminal vertices of G_j as follows. The label of the child t of s in $HT(\Gamma)$ corresponding to the nonterminal vertex (w, G_l) of G_j is (w, G_l).*

Given the above definition, we can naturally associate a hierarchy tree corresponding to each Γ_i, $1 \leq i \leq n$. We denote this tree by $HT(\Gamma_i)$. Note that each vertex v of $E(\Gamma)$ is either *an explicit vertex of G_n* or *the copy of some explicit vertex v' of G_j $(1 \leq j \leq n)$ in exactly one copy C_j^v of the graph $E(\Gamma_j)$ inserted during the construction of $E(\Gamma)$*. This enables us to assign v of $E(\Gamma)$ to the unique vertex n_v of the $HT(\Gamma)$ given by

1. if v is a terminal vertex of G_n, then n_v is the root of $HT(\Gamma)$, and

2. otherwise, v belongs to the node n_v that is the root of the hierarchy tree $HT(\Gamma_j)$, corresponding to C_j^v.

Given $HT(\Gamma)$, the *level number* of a node in $HT(\Gamma)$ is defined as the length of the path from the node to the root of the tree.

As noted in [35], L-specifications have the property that for each copy (instance) of a nonterminal, a complete boundary description has to be given. Thus if a nonterminal has a lot of pins, copying it is costly. Another property of the definition of L-specifications is that nonterminals are adjacent only to terminals. These properties ensure that the size of the "frontier" (or the number of neighbors) of any nonterminal is polynomial in the size of the specification. These properties weaken the

L-specifications with respect to other notions of hierarchy involving a substitution mechanism that entails implicit connections to pins at a cell boundary [11, 57]. As a result, regular structures such as grids cannot be specified using small L-specifications (see [32]). In contrast, the graph glueing model of Galperin [11] allows a hierarchical description of pins; thus the size of the frontier can be exponentially large. As a result, graphs such as grids can be represented using descriptions of logarithmic size. However, as demonstrated in [11, 32, 34, 35, 57], these properties seem to be a prerequisite for the construction of efficient exact algorithms for L-specified problems. As subsequent sections show, these restrictions are also necessary in part for devising efficient approximation algorithms for L-specified problems. The size of the frontier also has a significant impact on the complexity of several basic succinctly specified problems. For example, several basic NP-hard problems become PSPACE-hard when specified using L-specifications (see [37, 44]). In contrast, in a recent paper we show that these problems typically become NEXPTIME-hard when specified using the graph glueing specifications of [11] (see [45]).

By noting Definition 4.1, it follows that an L-specification is a restricted form of a context-free graph grammar. The substitution mechanism glues the pins of cells to neighbors of nonterminals representing these cells, as described in Definition 4.2. Such graph grammars are known as *hyperedge replacement systems* [15] or *cellular graph grammars* [38]. Two additional restrictions are imposed on cellular graph grammars to obtain L-specified graphs. First, for each nonterminal there is only one cell that can be substituted. Thus there are no *alternatives* for substitution. Second, the index of the substituted cell has to be *smaller* than the index of the cell in which the nonterminal occurs. The acyclicity condition together with the "no alternatives" condition implies that an L-specification defines a unique finite graph. We observe that $HT(\Gamma)$ is the parse tree of the unique graph generated by the context-free graph grammar Γ.

Example 2. Figure 4.1 depicts the L-specification $G = (G_1, G_2, G_3)$ and the associate hierarchy tree $HT(G)$. Figure 4.2 depicts the graph $E(G)$ specified by G. The correspondence between pins of G_j and neighbors of G_j in G_i, $j < i$, is clear by the positions of the vertices and the pins. □

4.1. Level-restricted specifications. We discuss level-restricted L-specifications now. This is also discussed in [40, 41].

DEFINITION 4.4. *An L-specification $\Gamma = (G_1, \ldots, G_n)$, $n \geq 1$, of a graph G is 1-level-restricted if for all edges (u, v) of $E(\Gamma)$, either*

1. *n_u and n_v are the same vertex of $HT(\Gamma)$, or*
2. *one of n_u or n_v is the parent of the other in $HT(\Gamma)$.*

Extending the above definition we can define k-level-restricted specifications. An L-specification $\Gamma = (G_1, \ldots, G_n)$, $n \geq 1$, of a graph $E(\Gamma)$ is *k-level-restricted*, if for all edges (u, v) of $E(\Gamma)$, either

1. n_u and n_v are the same vertex of $HT(\Gamma)$ or
2. one of n_u or n_v is an ancestor of the other in $HT(\Gamma)$ and the length of the path between n_u and n_v in $HT(\Gamma)$ is no more than k.

We note that for any fixed $k \geq 1$, k-level-restricted L-specifications can still lead to graphs that are exponentially large in the sizes of their specifications. Moreover, L-specifications (see [30, 31, 32]) for several practical designs are *k-level-restricted* for small values of k. (For example, it is easy to define a complete binary tree with $2^{\Omega(N)}$ nodes by a 1-level-restricted L-specification of size $O(N)$. Note however, that the specification depicted in Figure 4.1 is not 1-level-restricted.) For the rest of the

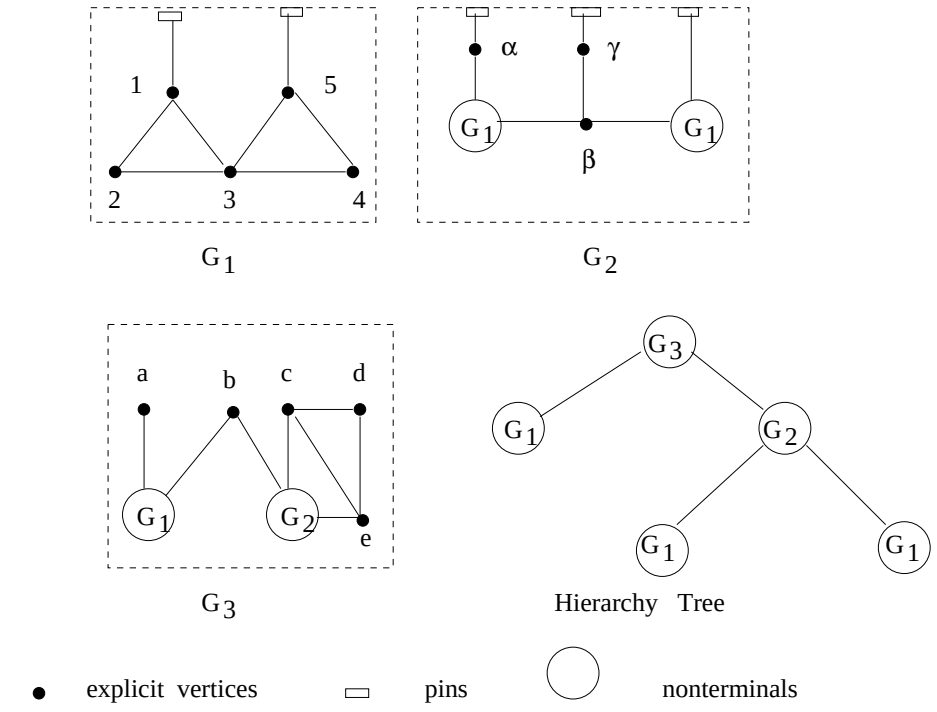


FIG. 4.1. *An L-specification G of a graph $E(G)$ and the associated hierarchy tree $HT(G)$. The mapping between the pins and its neighbors is clear by the relative positions of the pins and its neighbors.*

paper, given a problem Π specified using standard specifications, we use 1-L-Π to denote the problem specified using 1-level-restricted L-specifications and k-L-Π to denote the problem specified using k-level-restricted L-specifications.

5. 1-FPN-specifications. Next, we give the definition of one-dimensional periodic specifications due to Orlin [48], Wanke [58], and Höfting and Wanke [19]. For the rest of the paper $\mathbf{N}$ and $\mathbf{Z}$ denote the set of nonnegative integers and integers, respectively.

DEFINITION 5.1. *Let $G(V, E)$ (referred to as a static graph) be a finite directed graph such that each edge (u, v) has an associated nonnegative integral weight $t_{u,v}$. The undirected one-way infinite graph $G^{\infty}(V', E')$ is defined as follows:*

1. $V' = \{v(p) \mid v \in V$ *and* $p \in \mathbf{N}\}$,
2. $E' = \{(u(p), v(p + t_{u,v})) \mid (u, v) \in E$, $t_{u,v}$ *is the weight associated with the edge* (u, v) *and* $p \in \mathbf{N}\}$.

A one-dimensional periodic specification Γ *(referred to as* 1*-P-specification) is given by* $\Gamma = (G(V, E))$ *and specifies the graph* $G^{\infty}(V', E')$ *(referred to as a* 1*-P-specified graph).*

A 1*-P-specification* Γ *is said to be* narrow *or* 1-level-restricted *if* $\forall (u, v) \in E$, $t_{u,v} \in \{0, 1\}$. *This implies that* $\forall (u(p), v(q)) \in E'$, $|p - q| \leq 1$. *Similarly, a* 1*-P-specification is* k-narrow *or* k-level-restricted *if* $\forall (u, v) \in E$, $t_{u,v} \in \{0, 1, \ldots, k\}$.

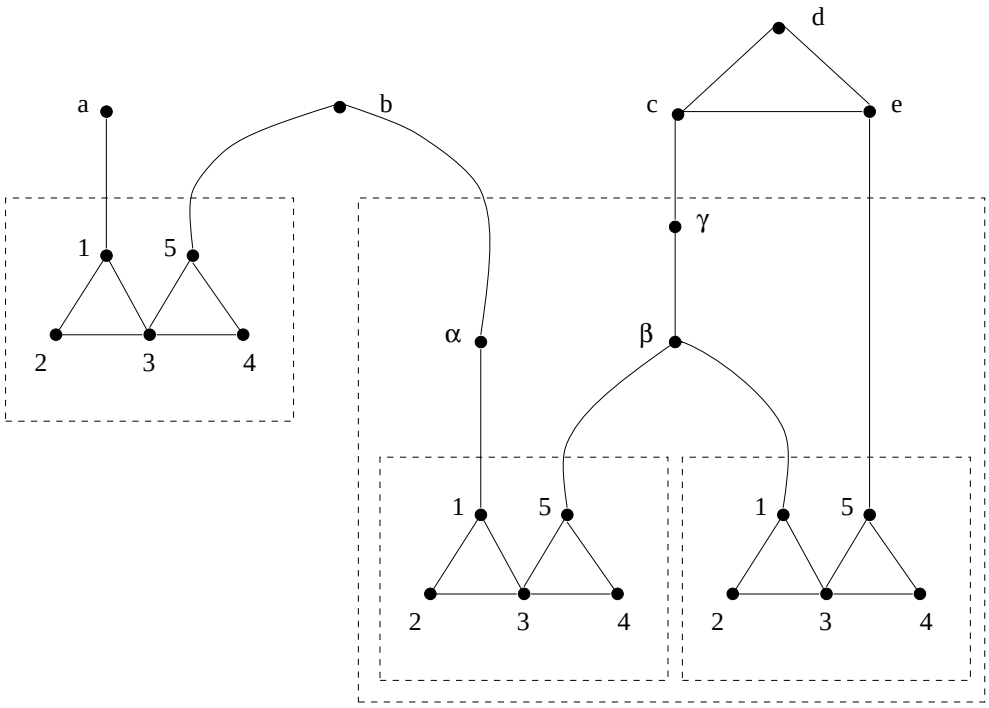


FIG. 4.2. *The graph $E(G)$ represented by G specified in Figure* 4.1.

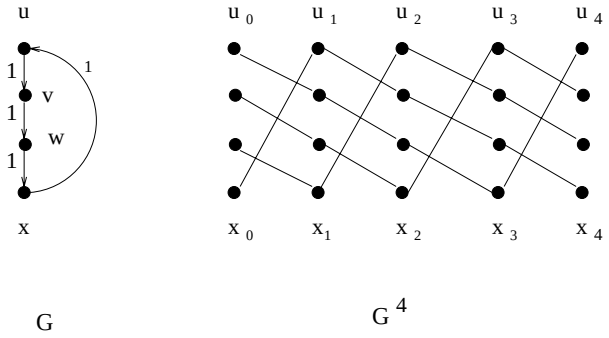


FIG. 4.3. *A static graph G, and the graph G^4 specified by the* 1*-FPN-specification* $\Gamma = (G, 4)$.

We note that if we replace **N** by **Z** in Definition 5.1, we obtain a *two-way* infinite periodically specified graph defined in Orlin [48]. It is sometimes useful to imagine a narrow periodically specified graph G^∞ as being obtained by placing a copy of the vertex set V at each integral point (also referred to as lattice point) on the X-axis (or the time line) and joining vertices placed on neighboring lattice points in the manner specified by the edges in E.

G^m is the subgraph of the infinite periodic graph G^∞ induced by the vertices associated with nonnegative lattice points less than or equal to m. Formally, we have the following definition.

DEFINITION 5.2. *Let $G(V, E)$ denote a static graph. Let $G^{\infty}(V', E')$ denote the one-way infinite* 1*-PN-specified graph as in Definition* 5.1. *Let $m \geq 0$ be an integer specified using binary numerals. Let $G^m(V^m, E^m)$ be a subgraph of $G^{\infty}(V', E')$ induced by the vertices $V^m = \{v(p) | v \in V$ and $0 \leq p \leq m\}$. A* 1*-FPN-specification is given by $\Gamma = (G(V, E), m)$ and specifies the graph G^m (referred to as a* 1*-FPN-specified graph).*

An example of a 1-FPN-specified graph appears in Figure 4.3. In [48], Orlin defined the concept of two-way infinite one-dimensional periodically specified 3CNF formulas and the associated 3SAT problem [12]. It is straightforward to restrict Orlin's definition along the lines of Definition 5.1 to define 1-FPN-specified satisfiability problems. As a consequence, we omit the definition here. (See [44, 48, 54] for formal definitions of periodically specified satisfiability problems.) We only give an example of a 1-FPN-specified 3CNF formula to illustrate the concept.

Example 3. Let $U = \{x_1, x_2, x_3\}$ be a set of static variables. Let C be a set of static clauses given by $(x_1(0) + x_2(0) + x_3(0)) \wedge (x_1(1) + x_3(0)) \wedge (x_3(1) + x_2(0))$. Let $F = (U, C, 3)$ be a 1-FPN-specification. Then F specifies the 3CNF formula $F^3(U^3, C^3)$ given by

$$(x_1(0) + x_2(0) + x_3(0)) \wedge (x_1(1) + x_3(0)) \wedge (x_3(1) + x_2(0)) \bigwedge$$

$$(x_1(1) + x_2(1) + x_3(1)) \wedge (x_1(2) + x_3(1)) \wedge (x_3(2) + x_2(1)) \bigwedge$$

$$(x_1(2) + x_2(2) + x_3(2)) \wedge (x_1(3) + x_3(2)) \wedge (x_3(3) + x_2(2)) \bigwedge$$

$$(x_1(3) + x_2(3) + x_3(3)).$$

6. Other preliminaries. Recall that a graph is said to be *planar* if it can be laid out in the plane in such a way that there are no crossovers of edges. For the rest of the paper, we use L-PL-Π, 1-L-PL-Π, and 1-FPN-PL-Π to denote the problem Π restricted to L-specified planar instances, 1-level-restricted L-specified planar instances, and 1-FPN-specified planar instances, respectively. As shown in Lengauer [35], given an L-specification Γ, there is a polynomial time algorithm to determine if $E(\Gamma)$ is planar. Similarly, as pointed out in [18], given a 1-FPN-specification Γ, there is a polynomial time algorithm to determine if $E(\Gamma)$ is planar. Thus for solving L- or 1-FPN-specified problems restricted to planar instances, we can assume without loss of generality that the inputs to our algorithms consist of planar instances.

Next, we define the problems MAX SAT(**S**). The definition is essentially an extension of the definition of SAT(**S**) given in Schaefer [56].

DEFINITION 6.1 (Schaefer [56]). *Let $\mathbf{S} = \{R_1, R_2, \ldots, R_m\}$ be a finite set of finite arity Boolean relations. (A Boolean relation is defined to be any subset of $\{0, 1\}^p$ for some integer $p \geq 1$. The integer p is called the* **arity** *of the relation.) An* **S-formula** *is a conjunction of clauses, each of the form $\hat{R}_i(\xi_1, \xi_2, \ldots)$, where $\xi_1, \xi_2, \ldots$ are distinct, unnegated variables whose number matches the arity of $R_i, i \in \{1, \ldots, m\}$, and $\hat{R}_i$ is the relation symbol representing the relation R_i. The* **S***-satisfiability problem is the problem of deciding whether a given* **S***-formula is satisfiable. Given an* **S***-formula F, the problem* MAX SAT(**S**) *is to determine the maximum number of simultaneously satisfiable clauses in F.*

As in Schaefer [56], given $\mathbf{S}$, $Rep(\mathbf{S})$ is the set of relations that are representable by existentially quantified $\mathbf{S}$-formulas with constants.

Recall from [39] that an $\mathbf{S}$-formula f is said to be planar if its associated bipartite graph is planar. The problem PL-3SAT [39] is the problem of determining if a given planar 3CNF formula is satisfiable. Lichtenstein [39] showed that the problem PL-3SAT is NP-complete.

Next, we define L-specified $\mathbf{S}$-formulas. Such formulas are built by defining larger $\mathbf{S}$-formulas in terms of smaller $\mathbf{S}$-formulas. Just as L-specifications of graphs can represent graphs that are exponentially larger than the specification, L-specified $\mathbf{S}$-formulas can specify formulas that are exponentially larger than the size of the specification.

DEFINITION 6.2. *An instance* $F = (F_1(X^1), \ldots, F_{n-1}(X^{n-1}), F_n(X^n))$ *of* L-SAT($\mathbf{S}$) *is of the form*

$$F_i(X^i) = \left(\bigwedge_{1 \le j \le l_i} F_{i_j}(X^i_j, Z^i_j) \right) \bigwedge f_i(X^i, Z^i)$$

for $1 \le i \le n$, *where* f_i *are* $\mathbf{S}$*-formulas,* $X^n = \phi$, X^i, X^i_j, Z^i, Z^i_j, $1 \le i \le n-1$, *are vectors of Boolean variables such that* $X^i_j \subseteq X^i$, $Z^i_j \subseteq Z^i$, $0 \le i_j < i$. *Thus,* F_1 *is just an* $\mathbf{S}$*-formula. An instance of* L-SAT($\mathbf{S}$) *specifies an* $\mathbf{S}$*-formula* $E(F)$, *which is obtained by expanding the* F_j, $2 \le j \le n$, *where the set of variables* Z*'s introduced in any expansion are considered distinct. The problem* L-SAT($\mathbf{S}$) *is to decide whether the formula* $E(F)$ *specified by* F *is satisfiable. The corresponding optimization problem denoted by* L-MAX-SAT($\mathbf{S}$) *is to find the maximum number of simultaneously satisfiable clauses in* $E(F)$.

Let n_i be the total number of variables used in F_i (i.e., $|X^i| + |Z^i|$) and let m_i be the total number of clauses in F_i. The size of F, denoted by $size(F)$, is equal to $\sum_{1 \le i \le n}(m_i n_i)$. Given a formula $E(F)$ specified by an L-specification F, $BG(E(F))$ denotes the bipartite graph associated with $E(F)$. We use $H[BG(E(F))]$ to denote the L-specification of $BG(E(F))$. It is easy to define level-restricted L-SAT($\mathbf{S}$) formulas along the lines of Definition 4.4. Hence we omit this definition here.

Example 4. Let $F = (F_1(x_1, x_2), F_2(x_3, x_4), F_3)$ be an instance of L-3SAT where each F_i is defined as follows:

$$F_1(x_1, x_2) = (x_1 + x_2 + z_1) \wedge (z_2 + z_3),$$

$$F_2(x_3, x_4) = F_1(x_3, z_4) \wedge F_1(z_4, z_5) \wedge (z_4 + z_5 + x_4),$$

$$F_3 = F_1(z_7, z_6) \wedge F_2(z_8, z_7).$$

The formula $E(F)$ denoted by F is $(z_7 + z_6 + z^1_1) \wedge (z^1_2 + z^1_3) \wedge (z_8 + z_4 + z^2_1) \wedge (z^2_2 + z^2_3) \wedge (z_4 + z_5 + z^3_1) \wedge (z^3_2 + z^3_3) \wedge (z_4 + z_5 + z_7)$.

We now extend the definition of PL-3SAT given in [39] to define the L-PL-3SAT.

DEFINITION 6.3. *The problem* L-PL-3SAT *is to decide whether the planar* 3*CNF formula* $E(F)$ *specified by an L-specification* F *is satisfiable. The corresponding optimization problem denoted by* L-PL-MAX-3SAT *is to find the maximum number of simultaneously satisfiable clauses in* $E(F)$.

Extensions of the above definition to 1-L-PL-3SAT,1-L-PL-MAX-3SAT, L-PL-SAT($\mathbf{S}$), L-PL-MAX-SAT($\mathbf{S}$), 1-L-PL-SAT($\mathbf{S}$), 1-L-PL-MAX-SAT($\mathbf{S}$), 1-FPN-PL-SAT($\mathbf{S}$), and 1-FPN-PL-MAX-SAT($\mathbf{S}$) are straightforward and are omitted.

Finally we state the following PSPACE-completeness results proved in a sequel paper [44]. The definitions of the problems mentioned in the following theorems can be found in [12].

THEOREM 6.4. *The following problems are PSPACE-complete for* 1*-level-restricted L-specified planar instances:* INDEPENDENT SET, VERTEX COVER, PARTITION INTO TRIANGLES, *and* SAT(**S**) *such that Rep*(**S**) *is the set of all finite arity Boolean relations.*

THEOREM 6.5. *The following problems are PSPACE-complete for* 1*-FPN-specified planar instances:* INDEPENDENT SET, VERTEX COVER, PARTITION INTO TRIANGLES, *and* SAT(**S**) *such that Rep*(**S**) *is the set of all finite arity Boolean relations.*

7. Approximation algorithms. The hardness results in Theorems 6.4 and 6.5 motivate the study of polynomial time approximation algorithms with good performance guarantees for these problems. We show that several basic combinatorial problems (including the ones in Theorems 6.4 and 6.5) have approximation algorithms with performance guarantees asymptotically equal to the best known performance guarantees, when instances are specified using standard specifications. As an immediate corollary, most of the problems shown to have PTASs in [3, 21] when instances are represented using standard specifications have PTASs when instances are specified by either k-level-restricted L-specifications or 1-FPN-specifications.

7.1. The basic technique: Partial expansion. We outline the basic technique behind the approximation algorithms for the 1-level-restricted L-specified problems. Consider one of the maximization problems Π in this paper. Let $\mathcal{A}$ be an approximation algorithm with performance guarantee $FBEST$ for Π when specified using standard specifications. Also, let $T(N)$ denote an increasing function that is an upper bound on the running time of $\mathcal{A}$ used to solve Π specified using standard specifications of size $O(N)$. Then, given a fixed $l \geq 1$, our approximation algorithm for 1-L-Π takes time $O(N \cdot T(N^{l+1}))$ and has a performance guarantee of $(\frac{l+1}{l}) \cdot FBEST$. Informally, the algorithm consists of $(l+1)$ iterations. During an iteration i we delete[2] all the explicit vertices which belong to nonterminals defined at level j, $j = i \mod (l+1)$. This breaks up the given hierarchy tree into a collection of disjoint trees. The algorithm finds a near-optimal solution for the vertex-induced subgraph[3] defined by each small tree and outputs the union of all these solutions as the solution for the problem Π. It is important to observe that the hierarchy tree can have an exponential number of nodes. Hence the deletion of nonterminals and the determination of near-optimal solutions for each subtree has to be done in such a manner that the whole process takes only polynomial time. This is achieved by observing that the subtrees can be divided into n distinct equivalence classes and that the number of subtrees in each equivalence class can be counted in polynomial time in the *size* of the specification.

We remark that our idea of dividing the graph into vertex (edge) disjoint subgraphs is similar to the technique used by Baker [3] for obtaining approximation schemes for planar graph problems.

7.2. Maximum independent set problem for 1-level-restricted L-specified planar graphs. We illustrate the technique by giving a PTAS for the MAXIMUM INDEPENDENT SET problem for 1-level-restricted L-specified planar graphs. The INDEPENDENT SET problem is defined as follows. Given a graph $G = (V, E)$ and a

[2]For the minimization problem, instead of deleting the vertices in the level, we consider the vertices as a part of both the subtrees.

[3]For a fixed l, the size of each subgraph is polynomial in the size of the specification.

positive integer $K \leq |V|$, is there an independent of size K or more for G, i.e., a subset $V' \subseteq V$ with $|V'| \geq K$ such that for each $u, v \in V'$ $(u, v) \notin E$? The optimization problem called the MAXIMUM INDEPENDENT SET PROBLEM (MIS) requires one to find an independent set of maximum size. In [40], we showed that given an L-specification that has edges between pins in the same cell, there is a polynomial time algorithm to construct a new L-specification such that there is no edge between pins in the *same* cell. Consequently, we assume without loss of generality that in the given L-specification there is no edge between two pins in the same nonterminal.

In the following description, we use $HIS(G_i)$ to denote the approximate independent set for the graph $E(\Gamma_i)$ obtained by our algorithm H-MIS. We also use F-MIS to denote the algorithm of Baker [3] for finding an approximate independent set in a planar graph specified using a standard specification. Before we discuss the details of the heuristic we define the concept of *partial expansion* of an L-specification. Recall that, for each nonterminal G_i there is a unique hierarchy tree $HT(G_i)$ rooted at G_i.

DEFINITION 7.1. *Let $\Gamma = (G_1, \ldots, G_n)$ be an L-specification of a graph $E(\Gamma)$. The partial expansion $PE(G_i^j)$ of the nonterminal G_i is constructed as follows:*

$j = 0$: $PE(G_i^j) = G_i -$ {all the explicit vertices defined in G_i}. (Thus the definition of $PE(G_i^j)$ now consists of a collection of the nonterminals and pins called in the definition of G_i.)

$j \geq 1$: Repeat the following step for each nonterminal G_r called by G_i: Insert a copy of $PE(G_r^{j-1})$ by identifying the lth pin of $PE(G_r^{j-1})$ with the node in G_i that is labeled (v, l). (Observe that the definition of $PE(G_i^j)$ consists of (i) *explicit vertices defined in all the nonterminals at depth r, $0 \leq r \leq j-1$, in $HT(G_i)$ and* (ii) *a* multiset *of nonterminals G_k, such that the nonterminal G_k occurs at depth $j+1$ in the hierarchy tree $HT(G_i)$.)*

Let $Ex(PE(G_i^j))$ denote the subgraph induced by the set of explicit vertices in the definition of $PE(G_i^j)$. Also let $V(E(\Gamma_i))$ denote the set of vertices in $E(\Gamma_i)$.

HEURISTIC H-MIS

- **Input:** A 1-level-restricted L-specification $\Gamma = (G_1, \ldots, G_n)$ of a planar graph G and an integer $l \geq 1$.

- **Output:** An L-specification of an independent set for $E(\Gamma)$ whose size is at least $(\frac{l}{l+1})^2$ times the size of an optimal independent set in $E(\Gamma)$.

1. **For** each $1 \leq i \leq l$, find a near-optimal independent set in $E(\Gamma_i)$ using F-MIS.
2. **For** each $l+1 \leq i \leq n-1$
   (a) Compute the partial expansion $PE(G_i^l)$ of G_i.
   (b) Find an independent set in the subgraph $Ex(PE(G_i^l))$ using heuristic F-MIS. Denote this by A_i^l.
   (c) Let $G_{i_1}, \ldots, G_{i_p}$ denote the multiset of nonterminals in $PE(G_i^l)$. Then the independent set for the whole graph for the iteration i denoted by $HIS(G_i)$ is given by

$$HIS(G_i) = A_i^l \cup \bigcup_{1 \leq r \leq p} HIS(G_{i_r}).$$

Remark. The explicit vertices in $PE(G_i^l)$ do not have an edge to any of the nonterminals $G_{i_1}, \ldots, G_{i_p}$. From this observation and the

definition of hierarchical specification, the independent set $HIS(G_i)$ can now be calculated as follows.

(d)
$$|HIS(G_i)| = |A_i^l| + \sum_{1 \le r \le p} |HIS(G_{i_r})|.$$

3. **For** each $0 \le i \le l$
   (a) Compute the partial expansion $PE(G_n^i)$ of G_n.
   (b) Find a near-optimal independent set of all the explicit vertices in $PE(G_n^i)$ using F-MIS. Denote this by A_n^i.
   (c) Let $G_{n_1}, \ldots, G_{n_p}$ denote the multiset of nonterminals in $PE(G_n^i)$. The independent set for the whole graph for the iteration i, denoted by $HIS_i(G_n)$, is given by
$$HIS_i(G_n) = A_n^i \cup \bigcup_{1 \le r \le p} HIS(G_{n_r}).$$
   Remark. By a remark similar to the one in step 2(c) of the algorithm, we have the following.

   (d)
$$|HIS_i(G_n)| = |A_n^i| + \sum_{1 \le r \le p} |HIS(G_{n_r})|.$$

4. The independent set $HIS(G)$ is the largest among all the independent sets $HIS_i(G_n)$ computed in step 3(c).
5. $|HIS(G)| = \max_{0 \le i \le l} |HIS_i(G_n)|$.

7.3. Analysis and performance guarantee. The correctness of H-MIS and the proof of its performance guarantee is based on the following intermediate results.

LEMMA 7.2. *The set $HIS(G)$ computed by the algorithm H-MIS in step* 4 *is an independent set.*

Proof. We first prove that the set for $1 \le i \le n-1$, $HIS(G_i)$, is an independent set. The proof is by induction on the depth of the hierarchy tree $HT(\Gamma)$.

Basis: If the depth is $\le l$, the proof follows by the correctness of algorithm F-MIS.

Induction: Assume that the lemma holds for all hierarchy trees of depth at most $m > l$. Consider a hierarchy tree of depth $m+1$. Step 2(c) of the algorithm computes a partial expansion $PE(G_i^l)$. This implies that the explicit vertices in $PE(G_i^l)$ do not have edges incident on the nonterminals in $PE(G_i^l)$. Thus, by the definition of 1-level-restricted L-specifications and partial expansion, it follows that the independent sets A_i^l and the sets $HIS(G_{i_r})$, $1 \le r \le p$, computed in steps 2(b) and 2(c) are disjoint. Also, the nonterminals in $PE(G_i^l)$ are at level $l+1$ in $HT(G_i)$ and have an associated hierarchy tree of depth $\le m$. Thus, by the induction hypothesis and the above stated observations, it follows that $HIS(G_m)$ computed in step 2(c) is an independent set. This completes the proof that $1 \le i \le n-1$, and $HIS(G_i)$ is an independent set.

A similar inductive argument proves that the set $HIS_i(G_n)$ computed in each iteration of step 3(c) is also an independent set. By step 4, we have that $HIS(G)$ is an independent set. □

LEMMA 7.3.

1. *In each iteration i, $l+1 \le i \le n-1$, of step* 2 *of algorithm H-MIS, all the explicit vertices in nonterminals at levels $j = l \mod (l+1)$ in the hierarchy tree $HT(G_i)$ are deleted.*

2. *In each iteration i of step* 3 *of algorithm H-MIS, all the explicit vertices in nonterminals at levels $j = i \mod (l+1)$ in the hierarchy tree $HT(G_n)$ are deleted.*

Proof of Part 1. Induction on the depth of the hierarchy tree associated with G_i.
Basis: If the depth is $l+1$, the proof follows directly by step 1 and the definition of partial expansion.
Induction: Assume that the lemma holds for all hierarchy trees of depth at most $m > (l+1)$. Consider a hierarchy tree of depth $m+1$. Step 2(c) of the algorithm computes the partial expansion $PE(G_i^l)$. This implies that all the explicit vertices at level l in the hierarchy tree $HT(G_i)$ were deleted. Each nonterminal occurring in the definition of $PE(G_i^l)$ is at level $l+1$ in $HT(G_i)$ and has an associated hierarchy tree of depth $\leq m$. The proof now follows by the induction hypothesis.

Proof of Part 2. Consider a hierarchy tree $HT(G_n)$. In iteration i of step 3 we compute $PE(G_n^i)$. This removes all the explicit vertices defined in nonterminals at level i. Also, by the definition of partial expansion it follows that all explicit vertices defined in nonterminals at levels 1 to i appear explicitly in the partially expanded graph. Therefore, the partially expanded graph now has nonterminals defined at level $i+1$ in the hierarchy tree $HT(G_n)$. The proof now follows as a consequence of part 1 of the lemma. □

Given the decomposition of $E(\Gamma)$ into a forest (as a result of removing explicit vertices in nonterminals at levels $j = i \mod (l+1)$ in the hierarchy tree $HT(G_n)$) we can associate a hierarchy tree with each of the subgraphs in the forest. Each such tree is a subtree of the original hierarchy tree $HT(\Gamma)$. Label each subtree by the type of nonterminal that is the root of the subtree. The proof of the following lemma is straightforward.

LEMMA 7.4.

1. *During each iteration i of step* 3 *of the algorithm H-MIS, the root of each subtree is labeled by one of the elements of the set $\{G_1, \ldots, G_{n-1}\}$.*

2. *For $1 \leq i \leq n$, let $H_1^i, \ldots, H_{r_i}^i$ be the set of graphs corresponding to the subtrees labeled G_i. Then for each i the graphs $H_1^i, \ldots, H_{r_i}^i$ are isomorphic.*

7.3.1. At least one good iteration exists. Next we prove that at least one iteration of step 3 has the property that the number of nodes of an optimal independent set that are deleted is a small fraction of the optimal independent set.

Let F_i denote the set of vertices obtained by deleting the explicit nodes in iteration i in step 3 of algorithm H-MIS. By Lemma 7.3 it follows that for each iteration i we did not consider the explicit vertices in levels $j_{i_1}, j_{i_2}, \ldots, j_{i_p}$ such that $1 \leq i_p \leq n$ and $j_{i_q} = i \mod (l+1)$, $1 \leq q \leq p$. Let S_i, $0 \leq i \leq l$, be the set of vertices not considered in iteration i of step 3. Let $IS(G_n)$ denote an optimum independent set in the graph $E(\Gamma)$. Let $IS_{opt}(S_i)$ denote the nodes in S_i included in the maximum independent set $IS(G_n)$.

LEMMA 7.5.

$$\max_{0 \leq i \leq l} |IS(F_i)| \geq \frac{l}{(l+1)} |IS(G_n)|.$$

Proof. By Lemma 7.3 and the algorithm H-MIS, it follows that

$$S_i \cap S_j = \phi, \quad \cup_{t=0}^{t=l} S_t = V(E(\Gamma)), \quad \text{and}$$

$$|IS_{opt}(S_0)| + |IS_{opt}(S_1)| + \cdots + |IS_{opt}(S_l)| = |IS(G_n)|.$$

Therefore,

$$\min_{0\le i\le l} |IS_{opt}(S_i)| \le |IS(G_n)|/(l+1),$$

$$\max_{0\le i\le l} |IS(F_i)| \ge |IS(G_n)| - \min_{0\le i\le l} |IS_{opt}(S_i)| \ge \frac{l}{(l+1)}|IS(G_n)|. \qquad \square$$

7.3.2. Performance guarantee and running time. We now prove that the above algorithm computes a near-optimal independent set. Given any $\epsilon > 0$, for some choice of positive integer l such that $(\frac{l}{l+1})^2 \ge (1-\epsilon)$, we show that algorithm H-MIS computes an independent set whose size is at least $(1-\epsilon)$ times the size of an optimal independent set. We first recall a similar lemma in [3] for planar graphs specified using standard specifications.

THEOREM 7.6 (see [3]). *For all fixed $l \ge 1$, given a planar graph G there is a linear time algorithm that computes an independent set $FIS(G)$ such that $|FIS(G)| \ge (\frac{l}{l+1}) \cdot |IS(G)|$, where $IS(G)$ denotes a maximum independent set in G.*

LEMMA 7.7. $|HIS_i(G_n)| \ge (\frac{l}{l+1}) \cdot |IS(F_i)|$.

Proof (induction on the number of nonterminals in the definition of Γ). The base case is fairly straightforward. Consider the induction step. By the definition of partial expansion it follows that

$$|IS(F_i)| = |IS(Ex(PE(G_n^i)))| + \sum_{1\le r\le p} |IS(PE(G_{n_r}))|.$$

From step 3(c) of the algorithm H-MIS we also know that

$$|HIS_i(G_n)| = |A_n^i| + \sum_{1\le r\le p} |HIS(G_{n_r})|.$$

From the induction hypothesis and Theorem 7.6 it follows that

$$A_n^i| \ge \left(\frac{l}{l+1}\right) \cdot |IS(Ex(PE(G_n^i)))| \quad \text{and}$$
$$|HIS(G_{n_r})| \ge \left(\frac{l}{l+1}\right) \cdot |IS(PE(G_{n_r}))|.$$

The lemma now follows. $\square$

THEOREM 7.8. $|HIS(G)| \ge (\frac{l}{l+1})^2 \cdot |IS(G)|$.

Proof. The proof follows from Lemma 7.5 and repeated application of Lemma 7.7. $\square$

THEOREM 7.9. *Let Γ be an L-specification with vertex number N. Given any $\epsilon > 0$, let $l \ge 1$ be an integer such that $(\frac{l}{l+1})^2 \ge (1-\epsilon)$. Then the approximation algorithm H-MIS runs in time $O(N^{l+2})$ and finds an independent set in $E(\Gamma)$ that is at least $(\frac{l}{l+1})^2$ times the size of an optimal independent set in $E(\Gamma)$.*

Proof. The performance guarantee follows by Theorem 7.8. Therefore, we only prove the claimed time bounds.

First consider step 1. Note that by Euler's formula, the number of edges in a planar graph with $O(N^l)$ vertices is also $O(N^l)$. Thus, the size of the graphs $E(\Gamma_i)$, $1 \le i \le l$, is $O(N^l)$. Hence the time required to compute the partial expansion is

$O(N^l)$. By Theorem 7.6, the time needed to compute an independent set in $E(\Gamma_i)$ is $O(N^l)$. Thus the total running time of step 1 is $O(N^l)$.

Next consider each iteration of step 2 of the algorithm H-MIS. Step 2(a) takes time $O(N^{l+1})$ since the size of the graph $PE(G_i^l)$ can be $O(N^{l+1})$. By Theorem 7.6, the time needed for executing step 2(b) is $O(N^l)$, since the number of nodes in $Ex(PE(G_i^l))$ can be $O(N^l)$. By Lemma 7.4, steps 2(c) and 2(d) together take time $O(N)$. Therefore, the total running time for executing one iteration of step 2 is $O(N^{l+1})$. Thus the total running time of step 2 is $nO(N^{l+1}) = O(N^{l+2})$.

A similar calculation shows that the total time needed to execute one iteration of step 3 is $O(N^{l+1})$. Thus the total time needed to execute step 3 is $(l+1)O(N^{l+1}) = O(N^{l+1})$.

Thus the total running time of the algorithm is $O(N^{l+2})$. □

7.4. L-specification of the solution and the query problem. In section 7.3, we showed how to solve the *size* problem for 1-L-MIS. We now discuss the *construction* problem. As noted in section 2.1 our algorithms for the four variants of the problem apply to the *same* independent set $HIS(G)$.

The L-specification of the solution can be easily constructed by slightly modifying the algorithm H-MIS as follows. Consider the iteration i of step 3 which gives the maximum independent set. Denote the iteration by i^*. The L-specification H of the solution consists of nonterminals $H_1, \ldots, H_n$. For $1 \leq j \leq n$ the explicit vertices of H_j are the explicit vertices in $PE(G_j^i)$ that are in the independent set. If $PE(G_j^l)$ calls nonterminals $G_{j_1}, \ldots, G_{j_m}$, then the nonterminal H_j calls the nonterminals $H_{j_1}, \ldots, H_{j_m}$. Observe that some of the nonterminals H_i may be redundant and these can be removed from the final specification. Given the L-specification of the solution, the query problem can be easily solved by examining if the given vertex occurs in the set of nodes specified by the L-specification of the solution. Given an L-specification of the solution, we can solve the output problem as follows. We traverse the hierarchy tree associated with H in a depth-first manner and output the vertices in the nonterminals visited during the traversal.

Observe that the only place we used planarity was to obtain a near-optimal solution for the maximum independent set problem for each partially expanded graph. In section 7.7 we use this observation to compute near-optimal solutions for problems for arbitrary 1-level-restricted L-specified graphs.

7.5. Other L-specified planar problems. Our technique can be applied to obtain efficient approximation algorithms for the following additional optimization problems: MINIMUM VERTEX COVER, MAXIMUM PARTITION INTO TRIANGLES, MINIMUM EDGE DOMINATING SET, MAXIMUM CUT, and MAX SAT($\mathbf{S}$) for any finite set of finite arity Boolean relations $\mathbf{S}$. The basic idea behind devising approximation schemes for these problems is similar to the ideas used to solve the MAXIMUM INDEPENDENT SET problem. Therefore, we only briefly discuss the method for MINIMUM VERTEX COVER and MAX SAT($\mathbf{S}$).

(1) MINIMUM VERTEX COVER. Given a graph $G = (V, E)$ and a positive integer $K \leq |V|$, is there a vertex cover of size K or less for G, i.e., a subset $V' \subseteq V$ with $|V'| \leq K$ such that for each edge $(u, v) \in E$ either u or v belongs to V'? The optimization problem requires one to find a vertex cover of minimum size.

In order to approximate the 1-L-PL-MINIMUM VERTEX COVER problem we do the following. Given an ϵ, we choose an l such that $(\frac{l+1}{l})^2 \leq (1+\epsilon)$. Next, we modify the definition of partial expansion so that instead of deleting the explicit vertices at levels

$(l+1)$ apart, we consider them in both sides of the partition. For each $0 \leq i < l$, the algorithm finds a near-optimal solution for the overlapping planar graphs induced by explicit vertices in levels $(jl+i)$ to $((j+1)l+i)$ for $j \geq 0$. The algorithm picks the best among all the vertex covers obtained for the different values of i. Let $OPT(G)$ denote an optimal vertex cover for G. The following lemma points out that the solution obtained is at most $(\frac{l+1}{l})^2$ times the optimal vertex cover. The proof of the lemma follows the same general argument given for the MAXIMUM INDEPENDENT SET problem.

LEMMA 7.10. *The size of the vertex cover obtained is no more than*

$$\left(\frac{l+1}{l}\right)^2 |OPT(G)|.$$

Proof. Consider an optimal solution $OPT(G)$ to the vertex cover problem. Then for some $0 \leq t < l$, at most $|OPT(G)|/l$ nodes in $OPT(G)$ are in levels congruent to t mod (l). Consider the iteration when the planar graphs are obtained by overlapping at levels congruent to t mod (l). Hence the size of an optimal vertex cover in this iteration is $(|OPT(G)| + |OPT(G)|/l)$. Now applying the known approximation scheme [3] for computing a near-optimal vertex cover for each of smaller subgraphs, we obtain a near-optimal vertex cover for the whole graph for iteration t. The size of the vertex cover obtained in this iteration is no more than $(|OPT(G)| + |OPT(G)|/l)\frac{l+1}{l}$. The reason is that the explicit vertices in the overlapping levels are counted twice, and the near-optimal vertex cover heuristic yields a vertex cover of size $(l+1)/l$ times the optimal vertex cover for each subgraph. Since the heuristic picks the minimum vertex over all values of i, it follows that the size of the vertex cover produced by the heuristic is no more than $(\frac{l+1}{l})^2|OPT(G)|$. □

(2) MAX SAT($\mathbf{S}$). In the following, we will assume that an instance F of 1-L-PL-MAX-SAT($\mathbf{S}$) is specified by $H[BG(E(F))]$ (i.e., the specification of the associated bipartite graph). The basic idea behind the approximation schemes for 1-L-MAX-PL-SAT($\mathbf{S}$) is as follows. For each i, $0 \leq i \leq 2l$, in increments of 2, we remove the explicitly defined clauses which are in levels j and $j+1$, such that $j = i \mod (l+1)$. This breaks the bipartite graph into a number of smaller bipartite graphs such that the formulas they denote do not share any variables or clauses. It is not difficult to modify the definition of partial expansion to obtain a decomposition as described above. Figure 7.1 shows how the variables in levels j and $j+1$ are redistributed. As in the case of the MAXIMUM INDEPENDENT SET problem, it is easy to see that there exists an iteration t, $0 \leq t \leq 2l$, such that at most $\frac{OPT}{(l+1)}$ clauses in OPT are deleted. Next, by the results in [21] the problem can be solved near-optimally for each smaller subformula. The union of the clauses satisfied for each small formula constitutes a solution for a given value of i. We pick the best solution for different values of i. This ensures that the best assignment to the variables over all values of i is at least $(\frac{l}{l+1})^2$ of an optimal assignment to the variables of the 1-L-PL-MAX-SAT($\mathbf{S}$) instance.

7.6. Extension to k-level-restricted instances. The technique used to solve various problems for 1-level-restricted L-specifications can be generalized to solve problems specified using k-level-restricted L-specifications. We only point out the essential differences. Again, for the purposes of illustration consider the problem k-L-PL-MIS. First note that we need to extend the definition of partial expansion so that we delete the explicit vertices in nonterminals at k consecutive levels. This implies that the time to compute $PE(G_i^l)$, $1 \leq i \leq n-1$, is $O(N^{l+k})$. The rest of the

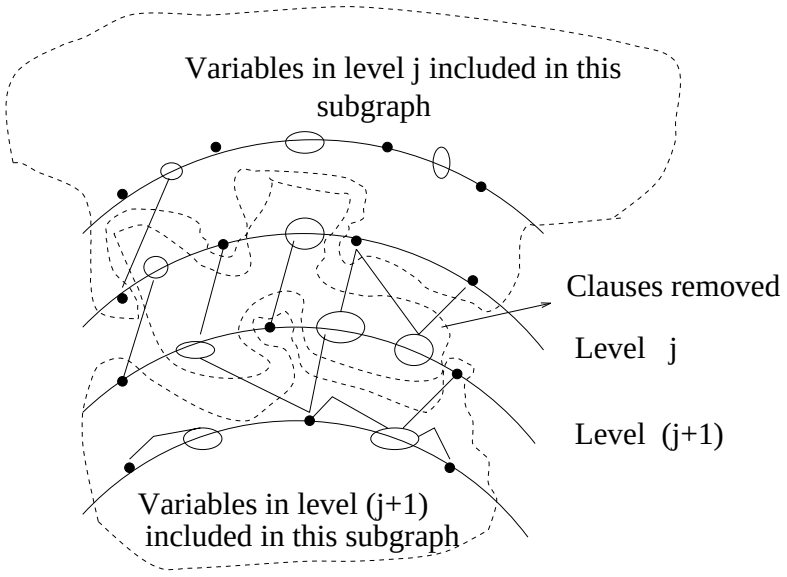


FIG. 7.1. *Basic idea behind the approximation algorithm for* 1-L-MAX-PL-SAT($\mathbf{S}$). *The black dots represent variables and the ellipses denote clauses. The figure depicts the set of clauses to be deleted and the redistribution of the variables.*

algorithm follows the same outline as that of H-MIS. The proof of correctness and the performance guarantee also follow similar arguments as in section 7.3. Thus the total running time of the algorithm is $O(N^{k+l+1})$ and its performance guarantee is $(\frac{l+1}{l})^2$. Hence we have the following theorem.

THEOREM 7.11. *For any fixed $k \geq 1$, there are PTASs for the problems* MAXIMUM INDEPENDENT SET, MINIMUM VERTEX COVER, MINIMUM EDGE DOMINATING SET, MAXIMUM PARTITION INTO TRIANGLES AND MAXIMUM CUT, *and* MAX SAT($\mathbf{S}$) *for each finite set of finite arity Boolean relations* $\mathbf{S}$, *when restricted to planar instances specified using k-level-restricted L-specifications.*

7.7. Extension to level-restricted arbitrary instances. Our results in sections 7.2–7.6 can be extended for problems on arbitrary graphs specified using k-level-restricted L-specifications. To do this, observe that to obtain the results in sections 7.2–7.6 we used planarity only to obtain approximation schemes for smaller subgraphs (formulas) obtained as a result of partial expansion. If the graphs were not planar we could use the best known approximation algorithms for solving the problem near-optimally and in turn get a performance guarantee which reflects this bound. For example, consider the problem 1-L-MAX-2SAT. Let $\epsilon > 0$ be the required performance guarantee. $l \geq 1$ is an integer satisfying the inequality $\frac{l}{l+1} \geq (1-\epsilon)$. For the problem MAX-2SAT, the recent work of Goemans and Williamson [14] provides an approximation algorithm with performance guarantee of 1.137. Using their algorithm as a subroutine to solve the small MAX-2SAT instances obtained as a result of partial expansion, we can devise an approximation algorithm for 1-L-MAX-2SAT with performance guarantee $\left(\frac{l+1}{l}\right)1.137$. A similar idea applies to other optimization problems considered. Again, it is easy to generalize our results for k-level-restricted L-specifications. Thus we have the following theorem.

Let Π be one of the following problems: MAXIMUM INDEPENDENT SET, MINIMUM VERTEX COVER, MINIMUM EDGE DOMINATING SET, MAXIMUM PARTITION INTO TRIANGLES, MAXIMUM CUT, and MAX-SAT($\mathbf{S}$) for finite set of Boolean relations $\mathbf{S}$, such that $Rep(\mathbf{S})$ is the set of all finite arity Boolean relations.[4]

THEOREM 7.12. *For all fixed $k \geq 1$, $\epsilon > 0$ and for all of the problems Π, there are polynomial time approximation algorithms with performance guarantee*[5] $(1+\epsilon) \cdot FBEST_{\Pi}$ *for problems Π, when specified using k-level-restricted L-specifications. Here $FBEST_{\Pi}$ denotes the best known performance guarantee of an algorithm for the problem Π for instances specified using standard specifications.*

Using the results of Arora et al. [2], Bellare, Goldreich, and Sudan [4] and Hunt, Marathe, and Stearns [22] we get the following theorem.

THEOREM 7.13. *Unless $\mathbf{P} = \mathbf{NP}$, the problems Π, when specified using k-level-restricted L-specifications, do not have PTASs.*

7.8. Approximation algorithms for 1-FPN-specified problems. Next, we briefly discuss how to extend our ideas developed in sections 7.2–7.7 in order to devise approximation schemes for several PSPACE-hard problems for 1-FPN-specified instances.

The basic idea is simple. Once again, we illustrate our ideas by describing our approximation algorithm for the problem 1-FPN-PL-MIS. Given a 1-FPN-specification $\Gamma = (G(V, E), m)$ of a planar graph G^m and an $\epsilon > 0$, we find the corresponding integer l that satisfies the inequality $(\frac{l}{l+1})^2 \geq (1 - \epsilon)$. For $0 \leq i \leq l$, we remove the vertices placed at the lattice points j such that $j = i \bmod (l + 1)$. This partitions the graph G^m into a number of smaller disjoint subgraphs, each induced by l consecutive lattice points.

Specifically, for a given i, let $l_p^i = \max\{0, (p-1)(l+1) + (i+1)\}$ and $r_p^i = \min\{m, p(l+1) + (i-1)\}$, where $0 \leq p \leq t_i$. Here $t_i = \lceil \frac{m-(i-1)}{(l+1)} \rceil$. Let the subgraph induced by vertices $v(j_p)$, where $l_p^i \leq j_p \leq r_p^i$, be denoted by $H(l_p^i, r_p^i)$. For a given $\epsilon > 0$, the graphs $H(l_p^i, r_p^i)$ are linear in the size of Γ. Figure 7.2 shows a schematic diagram of the vertices removed in a given iteration i. Next, we solve the MIS problem near-optimally on each of the subgraphs. This can be done by using the linear time algorithm stated in Theorem 7.6. The union of these independent sets is the independent set obtained in iteration i. The heuristic simply picks up the largest independent set obtained over all $l + 1$ iterations. By arguments similar to the ones we presented for approximating 1-L-PL-MIS (sections 7.2–7.4), it follows that the approximation algorithm has a performance guarantee of $(\frac{l+1}{l})^2$.

We note the following important point. If a near-optimal independent set were to be obtained for each subgraph $H(l_p^i, r_p^i)$, we would take an exponential amount of time in each iteration i. This is because $p = O(m)$. Hence we cannot afford to solve the problem explicitly for each subgraph. But observe that in each iteration i the subgraphs $H(l_p^i, r_p^i)$, $1 \leq p \leq \lceil \frac{m-(i-1)}{(l+1)} \rceil - 1$ are isomorphic. Hence we need to solve the MIS problem for the graphs $H(l_0^i, r_0^i), H(l_1^i, r_1^i)$, and $H(l_{t_i}^i, r_{t_i}^i)$, where $t_i = \lceil \frac{m-(i-1)}{(l+1)} \rceil$. Let $IS(H(l_p^i, r_p^i))$ denote the independent set obtained by the heuristic for the graph $H(l_p^i, r_p^i)$. Furthermore, let the approximate maximum independent set for the whole graph for a given iteration i be denoted by $IS(G^m(i))$. Then the size of $IS(G^m(i))$ is given by the following equation:

[4] Actually, our easiness results hold for all finite set of finite arity Boolean relations $\mathbf{S}$.

[5] For the sake of uniformity we assume that the performance guarantee is ≥ 1.

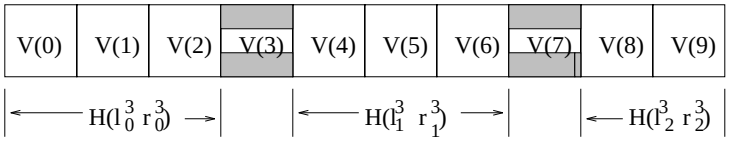


FIG. 7.2. *A schematic diagram showing the vertices to be removed in each iteration i while computing a near-optimal independent set for 1-FPN-specified planar graphs. In our example $i = 3$, $l + 1 = 4$, and $m = 9$. Each box represents a copy of the vertices in the original static graph. The shaded area represents the vertices that are removed.*

$$|IS(G^m(i))| = |IS(H(l_0^i, r_0^i))| + \left\lfloor \frac{m-(i-1)}{(l+1)} \right\rfloor |IS(H(l_1^i, r_1^i))| + |IS(H(l_{t_i}^i, r_{t_i}^i))|.$$

This completes the discussion of the approximation algorithm for 1-FPN-PL-MIS. By combining the above arguments along with those in sections 7.2–7.7, we can show that several other optimization problems can be approximated in a similar fashion. Again, it is easy to see that the technique extends to problems for arbitrary instances and also to problems for instances specified using k-narrow 1-FPN-specifications. Thus we have the following theorem.

THEOREM 7.14. *For all fixed $k \geq 1$, $\epsilon > 0$ and for all of the problems Π stated in section 7.7, there are polynomial time approximation algorithms with performance guarantee*[6] $(1+\epsilon) \cdot FBEST_\Pi$ *for problems Π, when specified using k-level-restricted 1-FPN-specifications. Here $FBEST_\Pi$ denotes the best known performance guarantee of an algorithm for the problem Π for instances specified using standard specifications.*

Observe that the technique used to devise approximation algorithms for problems restricted to k-narrow 1-FPN-specified instances is very similar to the technique used to devise approximation algorithms for k-level-restricted L-specified problems. But there are two important differences in the details of the algorithms.

1. In the case of algorithms for L-specified problems, the number of equivalence classes is $O(n)$, where n is the number of nonterminals. In contrast, the number of equivalence classes in the case of algorithms for 1-FPN-specified problems is only $O(1)$.

2. The size of the subgraphs for which the problem is solved near-optimally also differs significantly. Specifically, the number of explicit vertices in $PE(G_i^l)$ can be $O(N^l)$. Moreover, the time required to compute $PE(G_i^l)$ can be $O(N^{l+k})$. In contrast, the number of explicit vertices in each $H(l_p^i, r_p^i)$ is only $O(N)$ and the time required to construct each $H(l_p^i, r_p^i)$ is only $O(N)$. In both cases we use N to be the vertex number of the respective specifications Γ (N can be $O(size(\Gamma))$).

These important differences allow us to devise linear time approximation schemes for 1-FPN-specified problems.

8. Conclusions.

8.1. Summary. We have investigated the polynomial time approximability of several PSPACE-hard optimization problems for both L- and 1-FPN-specified instances. A general approach was given to obtain PTASs for several PSPACE-hard optimization problems for planar graphs specified using k-level-restricted L- or 1-FPN-specifications. We believe that the partial expansion technique can be used to obtain

[6] For the sake of uniformity we assume that the performance guarantee is ≥ 1.

efficient approximations for other problems specified using L- or 1-FPN-specifications as well as for problems specified using other succinct specifications.

In a companion paper [44], we investigate the decision complexity of various combinatorial problems specified using various kinds of L-specifications and 1-FPN-specifications. There we give a general method to obtain PSPACE-hard lower bounds for such problems, including the ones discussed here.

8.2. Open problems. We conclude with a list of open problems for future research.

1. Can we use the concept of probabilistically checkable debate systems [8, 9] to prove nonapproximability results for problems specified using arbitrary (not level-restricted) L-specifications?

Recently, Agarwal and Condon [1] have partially answered this question by showing that unless P = PSPACE, there is no PTAS for the problem L-MAX-3SAT. The result was proved by using the characterization of PSPACE in terms of random debate systems. In [22], we extended their result to hold for any L-MAX-SAT($\mathbf{S}$) such that *Rep*($\mathbf{S}$) denotes the set of all finite arity Boolean relations.

2. Recently, several researchers have considered logical definability of a number of optimization problems and defined appropriate classes such as MAX SNP, MAX Π_1, MAX NP, and MAX #P (cf. [25, 28, 51, 53]). All these researchers have assumed that the input is specified using standard specifications. What happens if the instances (finite or infinite) are specified succinctly ?

Some work has been done along these lines by Hirst and Harel [17]. Specifically, they considered infinite recursive versions of several NP optimization problems. They prove that some problems become highly undecidable (in terms of Turing degrees), while others remain on low levels of arithmetic hierarchy. As a corollary of their results they provide a method for proving (finitary) problems to be outside the syntactic class MAX NP and hence outside MAX SNP.

Acknowledgments. We thank the referees for invaluable comments that greatly improved the presentation. We also thank Anne Condon, Ashish Naik, Egon Wanke, Joan Feigenbaum, R. Ravi, S. S. Ravi, and Thomas Lengauer for many helpful conversations during the course of writing this paper.

REFERENCES

[1] S. Agarwal and A. Condon, *On approximation algorithms for hierarchical MAX-SAT*, in Proc. 10th Annual IEEE Conference on Structure in Complexity Theory, June 1995, pp. 181–190.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan, and M Szegedy, *Proof verification and hardness of approximation problems*, in Proc. 33rd IEEE Symposium on Foundations of Computer Science (FOCS), 1992, pp. 14–23.

[3] B.S. Baker, *Approximation algorithms for NP-complete problems on planar graphs*, J. Assoc. Comput. Mach., 41 (1994), pp. 153–180.

[4] M. Bellare, O. Goldreich, and M. Sudan, *Free bits, PCPs and non-approximability – towards tight results*, in Proc. 36rd IEEE Symposium on Foundations of Computer Science (FOCS'95), Oct. 1995, pp. 422–431.

[5] J.L. Bentley, T. Ottmann and P. Widmayer, *The complexity of manipulating hierarchically defined sets of rectangles*, in Advances in Computing Research 1, F.P. Preparata, ed., 1983, pp. 127–158.

[6] E. Cohen and N. Megiddo, *Recognizing properties of periodic graphs*, in Applied Geometry and Discrete Mathematics 4, The Victor Klee Festschrift, P. Gritzmann and B. Sturmfels, eds., ACM, New York, 1991, pp. 135–146.

[7] E. Cohen and N. Megiddo, *Strongly polynomial-time and NC algorithms for detecting cycles in dynamic graphs*, J. Assoc. Comput. Mach., 40 (1993), pp. 791–830.

[8] A. Condon, J. Feigenbaum, C. Lund, and P. Shor, *Probabilistically checkable debate systems and approximation algorithms for PSPACE-hard functions*, Chicago J. Theoret. Comput. Sci., 1995, Article 4 (http://www.cs.uchicago.edu/publications/cjtcs/articles/1995/4/contents.html). A preliminary version of the paper appears in Proc. 25th ACM Symposium on Theory of Computing (STOC), 1993, pp. 305–313.

[9] A. Condon, J. Feigenbaum, C. Lund, and P. Shor, *Random debaters and the hardness of approximating stochastic functions*, SIAM J. Comput., 26 (1997), pp. 369–400. A preliminary version of the paper appears in Proc. 9th IEEE Annual Conference on Structure in Complexity Theory, June 1994, pp. 280–293.

[10] D. Gale, *Transient flows in networks*, Michigan Math. J., 6 (1959), pp. 59–63.

[11] H. Galperin, *Succinct Representation of Graphs*, Ph.D. Thesis, Princeton University, Princeton, NJ, 1982.

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

[13] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of Software Engineering*, Prentice-Hall, Englewood Cliffs, NJ, 1991.

[14] M.X. Goemans and D.P. Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming,* J. Assoc. Comput. Mach., 42 (1995), pp. 1115–1145. A preliminary version appeared as .878 *approximation algorithms for MAX CUT and MAX* 2*SAT*, in Proc. 26th Annual ACM Symposium on Theory of Computing (STOC), 1994, pp. 422-431.

[15] A. Habel and H.J. Kreowski, *May we introduce to you: Hypergraph languages generated by hyperedge replacement*, in Proc. 13th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'87), Lecture Notes in Comput. Sci. 291, Springer-Verlag, New York, 1987, pp. 15–26.

[16] F.O. Hadlock, *Finding a maximum cut in a planar graph in polynomial time*, SIAM J. Comput., 4 (1975), pp. 221–225.

[17] T. Hirst and D. Harel, *Taking it to the limit: On infinite variants of NP-complete problems*, J. Comput. System Sci., 53 (1996), pp. 180–193.

[18] F. Höfting, T. Lengauer, and E. Wanke, *Processing of hierarchically defined graphs and graph families*, in Data Structures and Efficient Algorithms (Final Report on the DFG Special Joint Initiative), Lecture Notes in Comput. Sci. 594, Springer-Verlag, New York, 1992, pp. 44–69.

[19] F. Höfting and E. Wanke, *Minimum cost paths in periodic graphs*, SIAM J. Comput., 24 (1995), pp. 1051–1067.

[20] H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns, *A unified approach to approximation schemes for NP- and PSPACE-hard problems for geometric graphs*, in Proc. 2nd Annual European Symposium on Algorithms (ESA'94), 1994, pp. 424–435; J. Algorithms, 1998, to appear.

[21] H.B. Hunt III, M.V. Marathe, V. Radhakrishnan, D.J. Rosenkrantz, and R.E. Stearns, *Designing approximation schemes using L-reductions*, in Proc. 14th Annual Foundations of Software Technology and Theoretical Computer Science (FST &TCS), Madras, India, December 1994, pp. 342–353. A complete version of the paper titled *Parallel Approximation Schemes for Planar and Near-Planar Satisfiability and Graph Problems* is available as Technical Report No. LA-UR-96-2723, Los Alamos National Laboratory, Los Alamos, NM, 1996.

[22] H.B. Hunt III, M.V. Marathe, and R.E. Stearns, *Generalized CNF satisfiability problems and non-efficient approximability*, in Proc. 9th IEEE Conf. on Structure in Complexity Theory, 1994, pp. 356–366. A detailed version of the paper appears as SUNY-Albany Technical Report TR-95-27, Albany, NY, May 1995.

[23] K. Iwano and K. Steiglitz, *Testing for cycles in infinite graphs with periodic structure*, in Proc. 19th Annual ACM Symposium on Theory of Computing (STOC), 1987, pp. 46–53.

[24] K. Iwano and K. Steiglitz, *Planarity testing of doubly connected periodic infinite graphs*, Networks, 18 (1988), pp. 205–222.

[25] V. Kann, *On the Approximability of NP-Complete Optimization Problems*, Ph.D. Thesis, Dept. of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, Sweden, May 1992.

[26] R.M. Karp, R.E. Miller, and S. Winograd, *The organization of computations for uniform recurrence equations*, J. Assoc. Comput. Mach., 14 (1967), pp. 563–590.

[27] M. Kodialam and J.B. Orlin, *Recognizing strong connectivity in periodic graphs and its relation to integer programming*, in Proc. 2nd ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, Philadelphia, PA, 1991, pp. 131–135.
[28] P. G. Kolaitis and M.N. Thakur, *Logical definability of NP optimization problems*, Inform. and Comput., 115 (1994), pp. 321–353.
[29] K. R. Kosaraju and G.F. Sullivan, *Detecting cycles in dynamic graphs in polynomial time*, in Proc. 29th IEEE Symposium on Foundations of Computer Science (FOCS), 1988, pp. 398–406.
[30] T. Lengauer, *The complexity of compacting hierarchically specified layouts of integrated circuits*, in Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS), 1982, pp. 358–368.
[31] T. Lengauer, *Exploiting hierarchy in VLSI design*, in Proc. AWOC '86, Lecture Notes in Comput. Sci. 227, Springer-Verlag, New York, 1986, pp. 180–193.
[32] T. Lengauer and E. Wanke, *Efficient solutions for connectivity problems for hierarchically defined graphs*, SIAM J. Comput., 17 (1988), pp. 1063–1080.
[33] T. Lengauer and C. Weiner, *Efficient solutions for hierarchical systems of linear equations*, Computing, 39 (1987), pp. 111–132.
[34] T. Lengauer, *Efficient algorithms for finding minimum spanning forests of hierarchically defined graphs*, J. Algorithms, 8 (1987), pp. 260–284.
[35] T. Lengauer, *Hierarchical planarity testing*, J. Assoc. Comput. Mach., 36 (1989), pp. 474–509.
[36] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley, New York, 1990.
[37] T. Lengauer and K.W. Wagner, *The correlation between the complexities of non-hierarchical and hierarchical versions of graph problems*, J. Comput. System Sci., 44 (1992), pp. 63–93.
[38] T. Lengauer and E. Wanke, *Efficient decision procedures for graph properties on context-free graph languages*, J. Assoc. Comput. Mach., 40 (1993), pp. 368–393.
[39] D. Lichtenstein, *Planar formulae and their uses*, SIAM J. Comput., 11 (1982), pp. 329–343.
[40] M.V. Marathe H.B. Hunt III, and S.S. Ravi, *The complexity of approximating PSPACE-complete problems for hierarchical specifications*, Nordic J. Comput., 1 (1994), pp. 275–316.
[41] M.V. Marathe, V. Radhakrishnan, H.B. Hunt III, and S.S. Ravi, *Hierarchical specified unit disk graphs*, Theoret. Comput. Sci. 174 (1997), pp. 23–65. A preliminary version of the paper appears in Proc. 19th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '93), June 1993, pp. 21–32.
[42] M.V. Marathe, H.B. Hunt III, R.E. Stearns, and V. Radhakrishnan, *Approximation schemes for PSPACE-complete problems for succinct specifications*, in Proc. 26th Annual ACM Symposium on Theory of Computing (STOC), 1994, pp. 468–477.
[43] M.V. Marathe, *Complexity and Approximability of NP- and PSPACE-Hard Optimization Problems*, Ph.D. Thesis, Department of Computer Science, SUNY-Albany, Albany, NY, August 1994.
[44] M.V. Marathe, H.B. Hunt III, R.E. Stearns, and V. Radhakrishnan, *Complexity of hierarchically and 1-dimensional periodically specified problems*, in Proc. DIMACS Workshop on Satisfiability Problem: Theory and Applications, 1996. Also available as Technical Report LAUR-93-3348, Los Alamos National Laboratory, Los Alamos, NM, August, 1995.
[45] M.V. Marathe, H.B. Hunt III, D.E. Rosenkrantz, and R.E. Stearns, *Theory of Periodically Specified Problems: Complexity & Approximability*, Tech. Report LA-UR-96-1466, Los Alamos National Laboratory, Los Alamos, NM, 1996.
[46] J.O. McClain, L.J. Thomas, and J.B. Mazzola, *Operations Management*, Prentice-Hall, Englewood Cliffs, NJ, 1992.
[47] C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, MA, 1980.
[48] J.B. Orlin, *The Complexity of Dynamic/Periodic Languages and Optimization Problems*, Sloan Working Paper No. 1679-86, July 1985, Alfred P. Sloan School of Management, MIT, Cambridge, MA. A Preliminary version of the paper appears in Proc. 13th Annual ACM Symposium on Theory of Computing (STOC), 1981, pp. 218–227.
[49] J.B. Orlin, *Maximum convex cost dynamic network flows*, Math. Oper. Res., 9 (1984), pp. 190–206.
[50] J.B. Orlin, *Some problems on dynamic/periodic graphs*, in Progress in Combinatorial Optimization, Academic Press, New York, 1984, pp. 273–293.
[51] A. Panconesi and D. Ranjan, *Quantifiers and approximations*, Theoret. Comput. Sci., 107 (1993), pp. 145–163.
[52] C. Papadimitriou and M. Yannakakis, *A note on succinct representation of graphs*, Inform. and Comput., 71 (1986), pp. 181–185.

[53] C. Papadimitriou and M. Yannakakis, *Optimization, approximation and complexity classes*, J. Comput. System Sci., 43 (1991), pp. 425–440.
[54] C. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.
[55] D.J. Rosenkrantz and H.B. Hunt III, *The complexity of processing hierarchical specifications*, SIAM J. Comput., 22 (1993), pp. 627–649.
[56] T. Schaefer, *The complexity of satisfiability problems*, in Proc. 10th ACM Symposium on Theory of Computing (STOC), 1978, pp. 216–226.
[57] K.W. Wagner, *The complexity of problems concerning graphs with regularities*, in Proc. 11th Symposium on Math. Foundations of Computer Science (MFCS), Lecture Notes in Comput. Sci. 176, Springer-Verlag, New York, 1984, pp. 544–552.
[58] E. Wanke, *Paths and cycles in finite periodic graphs*, in Proc. 20th Symposium on Math. Foundations of Computer Science (MFCS), Lecture Notes in Comput. Sci. 711, Springer-Verlag, New York, 1993, pp. 751–760.
[59] M. Williams, *Efficient Processing of Hierarchical Graphs*, Tech. Report 90-06, Dept. of Computer Science, Iowa Sate University, Ames, IA. (Parts of the report appeared in WADS'89, pp. 563–576 and SWAT'90, pp. 320–331 coauthored with Fernandez-Baca.)
[60] M. Yannakakis, *On the approximation of maximum satisfiability*, J. Algorithms, 17 (1994), pp. 475–502.

SIAM J. COMPUT.
Vol. 27, No. 5, pp. 1262–1272, October 1998
© 1998 Society for Industrial and Applied Mathematics
003

APPROXIMATELY COUNTING HAMILTON PATHS AND CYCLES IN DENSE GRAPHS*

MARTIN DYER†, ALAN FRIEZE‡, AND MARK JERRUM§

Abstract. We describe fully polynomial randomized approximation schemes for the problems of determining the number of Hamilton paths and cycles in an n-vertex graph with minimum degree $(\frac{1}{2}+\alpha)n$, for any fixed $\alpha > 0$. We show that the exact counting problems are #P-complete. We also describe fully polynomial randomized approximation schemes for counting paths and cycles of all sizes in such graphs.

Key words. Hamilton cycles, fpras, dense

AMS subject classification. 68Q25

PII. S009753979426112X

1. Introduction. Combinatorial counting problems have a long history, even from the computational viewpoint. For example, the classical matrix-tree theorem provides a good algorithm for determining the number of trees in a graph. However, it seems that few interesting combinatorial structures possess good counting algorithms. This intuition was made precise by Valiant [21] using the class #P. He showed that many problems for which the decision counterpart is easy were nevertheless complete for this class. Since it is unlikely that #P = P, exact counting is apparently intractable for many natural problems. For example, Valiant [20] showed that 0-1 permanent evaluation and counting the number of bases of a (suitably presented) matroid [21] were #P-complete. Many other problems have since been added to this list, for example, volume computation for polyhedra [6], counting linear extensions of a partial order [3], and counting Eulerian orientations of a graph [17].

The hardness of most counting problems has led to an interest in *approximate* counting. The most fruitful approach in this respect has been *randomized approximation*. This is based on the idea of a fully polynomial randomized approximation scheme (*fpras*) due to Karp and Luby [15]. Thus, if N is the true value, we must determine an estimate $\widehat{N}$ such that for given $\varepsilon, \delta > 0$

$$\Pr\left(1/(1+\varepsilon) < \widehat{N}/N < (1+\varepsilon)\right) > 1-\delta,$$

in time polynomial in the size of the input, ε^{-1} and $\log(\delta^{-1})$. Examples of problems amenable to this type of approximation are dense 0-1 permanent evaluation [4, 12], matchings [12], volume computation [7], counting Eulerian orientations [17], counting linear extensions of a partial order [16], and computing the partition function for the ferromagnetic Ising model [13]. The algorithms in the papers cited use a *random*

* Received by the editors January 3, 1994; accepted for publication (in revised form) July 17, 1996; published electronically May 19, 1998.
http://www.siam.org/journals/sicomp/27-5/26112.html

† School of Computer Studies, University of Leeds, Leeds LS2 9JT, UK (dyer@dcs.leeds.ac.uk). This research was supported in part by Esprit Working Group "RAND."

‡ Mathematics Department, Carnegie Mellon University, Pittsburgh, PA 15213 (af1p@euler.math.cmu.edu). This research was supported in part by NSF grant 9002435.

§ Department of Computer Science, University of Edinburgh, The King's Buildings, Edinburgh EH9 3JZ, UK (mrj@dcs.ed.ac.uk). This research was supported in part by grant GR/F 90363 of the UK Science and Engineering Research Council and Esprit Working Group "RAND"; part of this work was done while the author was visiting the NEC Research Institute, Princeton, NJ.

walk to generate an almost uniform random solution to the problem (e.g., a random matching), and then apply multistage statistical sampling methods to obtain the desired estimate.

One obvious requirement for such approximate counting to be possible is that the associated decision problem be easy. In fact, it appears from experience that it must be "very easy" in order to have a realistic hope that a randomized approximation scheme can be found.

In this paper, we add further entries to the small but growing list of randomly approximable hard counting problems: that of counting the number of Hamilton paths and cycles in "dense" graphs. Let $G = (V, E)$ be a graph, where $V = \{v_1, v_2, \ldots, v_n\}$. Denote the degree of vertex v_i by d_i, for $i = 1, 2, \ldots, n$. We will say that G is *dense* if $\min_i d_i \geq (\frac{1}{2} + \alpha)n$, where $0 < \alpha \leq \frac{1}{2}$ is a fixed constant. Under these circumstances it is known [5] that G must contain a Hamilton cycle. Moreover, the proof of this fact is easily modified to give a simple polynomial-time algorithm for constructing such a Hamilton cycle. This algorithm, which uses edges whose existence is guaranteed by the pigeonhole principle to "patch together" disjoint cycles, provides the required easy decision procedure.

We consider here the natural but more difficult problems of counting the *number* of Hamilton paths and cycles in such graphs. We show in section 4 that these problems are in fact #P-complete, so exact counting is presumably intractable. More positively, our main results in sections 2 and 3 establish the existence of *fpras's* for these counting problems when $\alpha > 0$. We may observe that if the degree condition is relaxed to $\min_i d_i \geq (\frac{1}{2} - \alpha_n)n$ with $\alpha_n = \Omega(n^{\kappa-1})$ for any fixed $\kappa > 0$, then the question of the existence of any Hamilton path or cycle becomes NP-complete,[1] and approximate counting is NP-hard. Thus our results establish quite precisely the difficulty of the counting problem except in the region where α is close to zero. Section 5 extends the positive results of the earlier sections to cover self-avoiding paths and cycles of all lengths.

The natural approach given previous successes in this area is to try to find a rapidly mixing Markov chain with state space the set of Hamilton cycles of a given dense graph, and possibly its Hamilton paths as well. Earlier attempts with this approach have proved fruitless. Somewhat surprisingly, the key lies in the fact that in dense graphs, Hamilton cycles form a substantial fraction of the set of 2-factors, a 2-*factor* being defined as a set of vertex-disjoint cycles which together contain all vertices of G. This is not obvious a priori and the main technical difficulty in the approach lies in obtaining a good upper bound on the ratio of 2-factors to Hamilton cycles in a dense graph. A direct attack—relating the number of 2-factors with k cycles to the number with $k+1$ cycles—appears unworkable. Instead, we introduce a weight function on 2-factors that allows us to argue about the distribution of total weight as a function of the number of cycles. By a rather delicate analysis, we are able to show that the Hamilton cycles carry sufficient weight for our purpose. In summary, we prove the following theorem.

THEOREM 1.1. *If G is dense then there are fpras's for*

(a) *approximating its number of Hamilton cycles,*

[1] This is true even if we insist on G being k-connected for any $k = o(n)$. The construction is from Bollobás [2]. Start with an arbitrary graph G and add a clique C of size $m = n^{1/\kappa}$ and an independent set I of size $m - 1$, and then join every vertex in C to every other vertex, to produce a graph Γ. Then G has a Hamilton path if and only if Γ has a Hamilton cycle. Also, Γ contains a Hamilton path if and only if G contains two vertex-disjoint paths that cover all its vertices.

(b) *approximating its number of Hamilton paths,*
(c) *approximating its number of cycles of all sizes,*
(d) *approximating its number of paths of all sizes.*

2. Outline approach. Our approach to constructing an fpras for Hamilton cycles in a dense graph G is via a randomized reduction to sampling and estimating 2-*factors* in G. An *almost uniform sampler* for 2-factors in a graph is a randomized algorithm that takes as input a graph G and $\delta > 0$ and as output a 2-factor Z (a random variable) such that

$$1/(1+\delta)N \leq \Pr(Z = F) \leq (1+\delta)/N,$$

where F is any 2-factor in G and N is the total number of 2-factors. The sampler is said to be *fully polynomial* if it runs in time polynomial in the size of G and $\log \delta^{-1}$. Using known techniques, 2-factors in a dense graph G may be efficiently sampled, and their number estimated.

THEOREM 2.1. *There exist both a fully polynomial randomized approximation scheme and a fully polynomial almost uniform sampler for the set of* 2*-factors in a dense graph.*

This result follows immediately from Corollary 4.2 of Jerrum and Sinclair [14], as will become clear once the notation used there has been explained. For convenience, the corollary in question is repeated below as Proposition 2.2.

PROPOSITION 2.2. *There exists a fully polynomial almost uniform sampler for* $\mathcal{G}(\mathbf{d}, X)$ *and a fully polynomial randomized approximation scheme for* $|\mathcal{G}(\mathbf{d}, X)|$*, provided the pair* $(\mathbf{d}, X)$ *satisfies* $e(\mathbf{d}) > d_{\max}(d_{\max} + x_{\max} - 1)$.

In Proposition 2.2, $\mathbf{d} = (d_1, \ldots, d_n)$ stands for a degree sequence on $V = \{v_1, v_2, \ldots, v_n\}$, and $X \subseteq V^{(2)}$ for the edge set of an "excluded" graph on vertex set V. The notation $\mathcal{G}(\mathbf{d}, X)$ stands for the set of graphs on vertex set V that have degree sequence $\mathbf{d}$ and avoid all edges in X. Finally, $e(\mathbf{d})$ is the number of edges in any graph with $\mathbf{d}$ as degree sequence, $d_{\max}$ is the largest component of $\mathbf{d}$, and $x_{\max}$ is the largest degree of any vertex in the excluded graph (V, X).

Proof of Proposition 2.2 (sketch). Our aim here is merely to indicate the algorithmic techniques used to sample from, and estimate the size of, $\mathcal{G}(\mathbf{d}, X)$. For a full proof, the reader is directed to [14].

Using a reduction due to Tutte [19], a graph Γ is constructed whose perfect matchings are in (constant) many-one correspondence with elements of $\mathcal{G}(\mathbf{d}, X)$. An algorithm of Jerrum and Sinclair [12], based on the simulation of a rapidly mixing Markov chain, is then used to sample or estimate the number of perfect matchings in Γ, as required. For this algorithm to be applicable, we require that Γ satisfy a certain condition; it is this condition, translated back through the reduction to the pair $(\mathbf{d}, X)$, that gives rise to the condition $e(\mathbf{d}) > d_{\max}(d_{\max} + x_{\max} - 1)$. □

Proof of Theorem 2.1. The set of 2-factors in a graph $G = (V, E)$ is equal to $\mathcal{G}(\mathbf{d}, X)$, where $\mathbf{d} = (2, 2, \ldots, 2)$, and $X = V^{(2)} - E$ is the complementary edge set to E. The result now follows from Proposition 2.2, since, for a dense G and n sufficiently large, $d_{\max} = 2$, $x_{\max} < \frac{1}{2}n - 1$, and $d_{\max}(d_{\max} + x_{\max} - 1) < n = e(\mathbf{d})$. □

Given Theorem 2.1, the reduction from Hamilton cycles to perfect matchings is easy to describe. We estimate first the number of 2-factors in G, and then the number of Hamilton cycles by standard sampling methods as a proportion of the number of 2-factors. Both counting and sampling phases run in polynomial time, by Theorem 2.1, provided only that G is dense. For the sampling phase to produce an

accurate estimate, it is necessary that the ratio of 2-factors to Hamilton cycles in G not be too large. This will be established in section 3.

We remark that it would be sufficient to be able to generate a random Hamilton cycle. We could then proceed alternatively by adding one edge at a time, giving a sequence of $M = O(n^2)$ graphs $G = G_0, G_1, \ldots, G_M = K_n$. We could then estimate the ratio of the number of Hamilton cycles in G_{i-1} to those in G_i for $i = 1, 2, \ldots, M$. The degree conditions can be used to show that each of these ratios is not too small and hence can be estimated efficiently. (This is similar to an idea in [4].)

The method of using random 2-factors to generate random Hamilton cycles was previously used by Frieze and Suen [9] in the context of *random digraphs* and more recently by Frieze, Jerrum, and Molloy [8] with regard to random regular graphs. It is interesting that the same method should be successful here also. It raises the intriguing possibility of using existing approaches to other random graph problems to guide the design of new randomized algorithms for restricted versions of the corresponding deterministic problem.

3. Many 2-factors are Hamiltonian. Let n be a natural number and $\beta = 10/\alpha^2$. Let $k_0 = \lfloor \beta \ln n \rfloor$, and for $1 \le k \le n$, define $g(k) = n^\beta k!(\beta \ln n)^{-k}$, and

$$f(k) = \begin{cases} g(k), & \text{if } k \le k_0, \\ g(k_0), & \text{otherwise.} \end{cases}$$

LEMMA 3.1. *Let f be the function defined above. Then*
(a) *f is nonincreasing and satisfies*

$$\min\{f(k-1), f(k-2)\} = f(k-1) \ge (\beta \ln n) k^{-1} f(k);$$

(b) *$f(k) \ge 1$, for all k.*

Proof. Observe that g is unimodal and that k_0 is the value of k minimizing $g(k)$; it follows that f is nonincreasing. When $k \le k_0$, we have $f(k-1) = g(k-1) = (\beta \ln n) k^{-1} g(k) = (\beta \ln n) k^{-1} f(k)$; otherwise, $f(k-1) = g(k_0) = f(k) \ge (\beta \ln n) k^{-1} f(k)$. In either case, the inequality in part (a) of the lemma holds.

Part (b) of the lemma follows from the chain of inequalities

$$\frac{1}{f(k)} \le \frac{1}{g(k_0)} \le \frac{(\beta \ln n)^{k_0}}{n^\beta k_0!} \le n^{-\beta} \sum_{k=0}^{\infty} \frac{(\beta \ln n)^k}{k!} = n^{-\beta} \exp(\beta \ln n) = 1. \qquad \square$$

LEMMA 3.2. *Suppose α is constant greater than 0. Let $G = (V, E)$ be an undirected graph of order n and minimum degree $(\frac{1}{2} + \alpha)n$. Then the number of 2-factors in G exceeds the number of Hamilton cycles by at most a polynomial (in n) factor, the degree of the polynomial depending only on α.*

Proof. For $1 \le k \le \lfloor n/3 \rfloor$, let Φ_k be the set of all 2-factors in G containing exactly k cycles, and let $\Phi = \cup_k \Phi_k$ be the set of all 2-factors. Define

$$\Psi = \big\{(F, F') : F \in \Phi_k, F' \in \Phi_{k'}, k' < k, \\ \text{and } F \oplus F' \cong C_6\big\},$$

where $\oplus$ denotes symmetric difference and C_6 is the cycle on six vertices. Observe that (Φ, Ψ) is an acyclic directed graph; let us agree to call its component parts *nodes* and *arcs* to avoid confusion with the vertices and edges of G. Observe also that if $(F, F') \in \Psi$ is an arc, then F' can be obtained from F by deleting three edges and

adding three others, and that this operation can decrease the number of cycles by at most two. Thus every arc $(F, F') \in \Psi$ is directed from a node F in some Φ_k to a node F' in Φ_{k-1} or Φ_{k-2}.

Our proof strategy is to define a positive weight function on the arc set Ψ such that the total weight of arcs leaving each node (2-factor) $F \in \Phi \setminus \Phi_1$ is at least one greater than the total weight of arcs entering F. This will imply that the total weight of arcs entering Φ_1 is an upper bound on the number of non-Hamilton 2-factors in G, and that the maximum total weight of arcs entering a single node in Φ_1 is an upper bound on the ratio $|\Phi \setminus \Phi_1|/|\Phi_1|$.

The weight function $w : \Psi \to \mathbb{R}^+$ we employ is defined as follows. For any arc (F, F') with $F' \in \Phi_k$: if the 2-factor F' is obtained from F by coalescing two cycles of lengths l_1 and l_2 into a single cycle of length $l_1 + l_2$, then $w(F, F') = (l_1^{-1} + l_2^{-1})f(k)$; if F' results from coalescing three cycles of length l_1, l_2, and l_3 into a single one of length $l_1 + l_2 + l_3$, then $w(F, F') = (l_1^{-1} + l_2^{-1} + l_3^{-1})f(k)$.

Let $F \in \Phi_k$ be a 2-factor with $k > 1$ cycles $\gamma_1, \gamma_2, \ldots, \gamma_k$, of lengths $n_1, n_2, \ldots, n_k$. We proceed to bound from below the total weight of arcs *leaving* F. For this purpose imagine that the cycles $\gamma_1, \gamma_2, \ldots, \gamma_k$ are oriented in some way, so that we can speak of each oriented edge (u, u') in some cycle γ_i as being "forward" or "backward." Since we are interested in obtaining a *lower* bound, it is enough to consider only arcs (F, F^+) from F of a certain kind: namely, those for which the 6-cycle $\gamma = F \oplus F^+$ is of the form $\gamma = (x, x', y, y', z, z')$, where $(x, x') \in F$ is a forward cycle edge, $(y, y') \in F$ is a forward edge in a cycle distinct from the first, and $(z, z') \in F$ is a backward cycle edge. The edge (z, z') may be in the same cycle as either (x, x') or (y, y'), or in a third cycle. Observe that (x', y), (y', z), and (z', x) must necessarily be edges of F^+. It is routine to check that any cycle $\gamma = (x, x', y, y', z, z')$ satisfying the above constraints does correspond to a valid arc from F. The fact that (z, z') is oriented in the opposite sense to (x, x') and (y, y') plays a crucial role in ensuring that the number of cycles decreases in the passage to F^+ when only two cycles are involved.

First, we estimate the number of cycles γ for which (x, x') is contained in a particular cycle γ_i of F. We might say that γ is *rooted* at γ_i. Assume, for a moment, that the vertices x, x', y, y' have already been chosen. There are at least $(\frac{1}{2} + \alpha)n - 5$ ways to extend the path (x, x', y, y'), first to z and then to z', which are consistent with the rules given above; let Z' be the set of all vertices z' so reachable. Denote by $G(x)$ the set of vertices adjacent to x. The number of ways of completing the path (x, x', y, y') to a valid 6-cycle is at least

$$\begin{aligned} |G(x) \cap Z'| &\geq |G(x)| + |Z'| - n \\ &\geq (\tfrac{1}{2} + \alpha)n + [(\tfrac{1}{2} + \alpha)n - 5] - n \\ &= 2\alpha n - 5 \\ &\geq \alpha n, \end{aligned}$$

for n sufficiently large. A lower bound on the number of 6-cycles γ rooted at γ_i now follows easily: there are n_i choices for (x, x'); then at least $(\frac{1}{2} + \alpha)n - n_i$ choices for (y, y'); and finally—as we have just argued—at least αn ways to complete the cycle. Thus the total number of 6-cycles rooted at γ_i is at least $\alpha n n_i[(\frac{1}{2} + \alpha)n - n_i]$.

We are now poised to bound the total weight of arcs leaving F. Each arc (F, F^+) defined by a cycle γ rooted at γ_i has weight at least $n_i^{-1} \min\{f(k-1), f(k-2)\}$, which, by Lemma 3.1, is bounded below by $(\beta \ln n)(kn_i)^{-1} f(k)$. Thus the total weight of

arcs leaving F is bounded as follows:

$$\sum_{F^+:(F,F^+)\in\Psi} w(F,F^+) \geq \sum_{i=1}^{k} \alpha n n_i[(\tfrac{1}{2}+\alpha)n - n_i]\frac{(\beta \ln n)f(k)}{kn_i} \tag{1}$$

$$= \alpha n^2[(\tfrac{1}{2}+\alpha)k - 1]\frac{(\beta \ln n)f(k)}{k}$$

$$\geq \alpha^2 \beta\, f(k) n^2 \ln n$$

$$\geq 10 f(k) n^2 \ln n, \tag{2}$$

where we have used the fact that $k \geq 2$. Note that the presence of a unique backward edge, namely (z, z'), ensures that each cycle γ has a distinguishable root, and hence that the arcs (F, F^+) were not overcounted in summation (1).

We now turn to the corresponding *upper* bound on the total weight of arcs $(F^-, F) \in \Psi$ *entering* F. It is straightforward to verify that the cycle $\gamma = (x, x', y, y', z, z') = F^- \oplus F$ must contain three edges—(x, x'), (y, y') and (z, z')—from a single cycle γ_i of F, the remaining edges coming from F^-. The labeling of vertices in γ can be made canonical in the following way: assume an ordering on vertices in V, and assign label x to the smallest vertex. The condition $(x, x') \in F$ uniquely identifies vertex x', and the labeling of the other vertices in the cycle γ follows.

Removing the three edges (x, x'), (y, y'), and (z, z') from γ_i leaves a triple of simple paths of lengths (say) $a-1$, $b-1$, and $c-1$: these lengths correspond (resp.) to the segment containing x, the segment containing x', and the remaining segment. Going round the cycle γ_i, starting at x' and ending at x, the vertices x, x', y, y', z, z' may appear in one of eight possible sequences:

$$x', y', y, z', z, x;$$
$$x', z, z', y, y', x;$$
$$x', z, z', y', y, x;$$
$$x', z', z, y, y', x;$$
$$x', y', y, z, z', x;$$
$$x', y, y', z', z, x;$$
$$x', z', z, y', y, x;$$
$$x', y, y', z, z', x.$$

For a given triple of lengths (a, b, c), each of the above sequences corresponds to at most n_i possible choices for the edges (x, x'), (y, y'), and (z, z'), yielding a maximum of $8n_i$ in total. To see this, observe that the edge (x, x') may be chosen in n_i ways (minimality of x fixes the orientation of the edge), and that the choice of (x, x') combined with the information provided by the sequence completely determines the triple of edges.

The eight sequences divide into five possible cases, as the first four sequences lead to equivalent outcomes (covered by case 1 below). Taken in order, the five cases are:

1. for at most $4n_i$ of the choices for the edges (x, x'), (y, y'), and (z, z'), $\gamma_i \oplus \gamma$ is a single cycle;
2. for at most n_i choices, $\gamma_i \oplus \gamma$ is a pair of cycles of lengths a and $b + c$;
3. for at most n_i choices, $\gamma_i \oplus \gamma$ is a pair of cycles of lengths b and $a + c$;
4. for at most n_i choices, $\gamma_i \oplus \gamma$ is a pair of cycles of lengths c and $a + b$;
5. for at most n_i choices, $\gamma_i \oplus \gamma$ is a triple of cycles of lengths a, b, and c.

The first case does not yield an arc (F^-, F), since the number of cycles does not decrease when passing from $F^- = F \oplus \gamma$ to F, but the other four cases do have to be reckoned with.

The total weight of arcs entering F can be bounded above as follows:

$$\begin{aligned}\sum_{F^-:(F^-,F)\in\Psi} w(F^-,F) &\le \sum_{i=1}^{k} n_i f(k) \sum_{\substack{a,b,c\ge 1\\ a+b+c=n_i}} \left[\left(\frac{1}{a}+\frac{1}{b}+\frac{1}{c}\right)+\right.\\ &\qquad \left.\left(\frac{1}{a}+\frac{1}{b+c}\right)+\left(\frac{1}{b}+\frac{1}{a+c}\right)+\left(\frac{1}{c}+\frac{1}{a+b}\right)\right]\\ &= \sum_{i=1}^{k} n_i f(k) \sum_{\substack{a,b,c\ge 1\\ a+b+c=n_i}} \left[\frac{6}{a}+\frac{3}{b+c}\right]\\ &\le \sum_{i=1}^{k} n_i f(k) n \sum_{a=1}^{n_i-1} \left[\frac{6}{a}+\frac{3}{n_i-a}\right]\\ &\le 9f(k)n^2 H_n, \qquad (3)\end{aligned}$$

where $H_n = \sum_{i=1}^{n} i^{-1} \le \ln n + 1$ is the nth harmonic number [10, eq. (6.60)]. Combining inequalities (2) and (3), we have

$$\begin{aligned}\sum_{F^+:(F,F^+)\in\Psi} w(F,F^+) \;-\; \sum_{F^-:(F^-,F)\in\Psi} w(F^-,F) &\ge 10f(k)n^2\ln n - 9f(k)n^2 H_n\\ &\ge f(k)n^2(\ln n - 9)\\ &\ge n^2(\ln n - 9),\end{aligned}$$

where the final inequality is by Lemma 3.1. Thus the total weight of arcs leaving F exceeds the total weight of arcs entering by at least 1, provided n is sufficiently large. The number of non-Hamilton 2-factors $|\Phi\setminus\Phi_1|$ is bounded above by the total weight of arcs entering Φ_1, which in turn is bounded—see inequality (3)—by $|\Phi_1|\times 9f(1)n^2H_n = |\Phi_1|\times O(n^{2+\beta})$. This establishes the lemma. □

4. Exact counting is #P-complete. Let #HC (resp., #HP) be the problem of counting the number of Hamilton cycles (resp., paths) in an undirected graph. It is known [21, 18] that #HC is #P-complete, and it follows, by an easy reduction, that #HP is also #P-complete.

THEOREM 4.1. *Both #HC and #HP are #P-complete when restricted to graphs G of minimum degree at least $(1-\alpha)n$, where n is the number of vertices in G, and $\alpha > 0$.*

Proof. We first present a Turing reduction from #HP to #HP such that all the target instances of #HP satisfy the required minimum degree condition. Let $G = (V, E)$ be an undirected graph of order n with vertex set V and edge set E, considered as an instance of #HP. A typical target instance of #HP is a graph G_t constructed from G by forming the disjoint union of G with the complete graph K_t on $t \ge n$ vertices, and connecting every vertex in G with every vertex in K_t.

Assume $t \ge 3$. For $1 \le k \le n$, denote by P_k the set of all covers of G by k vertex-disjoint oriented paths, where paths of length 0 are allowed. Each oriented Hamilton path P in G_t induces an element of $\cup_k P_k$ by restriction to G. Conversely, each element of P_k may be extended in precisely $t!\binom{t+1}{k}k! = (t+1)!\binom{t}{k-1}(k-1)!$ ways to an oriented Hamilton path in G_t: the vertices in K_t may be visited in $t!$ orders; there are $\binom{t+1}{k}$ ways to choose k positions in that order during which excursions to G can be made, including the two positions prior to and following the order, and $k!$ ways

to match those positions to the k oriented paths covering G. Thus the number p_t of oriented Hamilton paths in G_t can be expressed as the sum

$$p_t = \sum_{k=1}^{n} (t+1)! \binom{t}{k-1} (k-1)! |P_k|.$$

Note that p_t may be evaluated by one call to an oracle for #HP, since the number of oriented Hamilton paths is twice the number of unoriented paths. Using n such calls we may evaluate p_t for $t = t_0 + j$ and $j = 1, 2, \ldots, n$, where $t_0 = \lceil \alpha^{-1} n \rceil$ is chosen sufficiently large that every graph G_t with $t > t_0$ satisfies the minimum degree constraint. Recovering the values $\{(k-1)!|P_k| : 1 \le k \le n\}$ from $\{p_{t_0+j}/(t_0+j+1)! : 1 \le j \le n\}$ amounts to inverting the matrix

$$A = (A_{jk}) = \left(\binom{t_0 + j}{k-1} : 1 \le j, k \le n \right),$$

which may be expressed as the product $A = LU$ of a lower triangular matrix $L = (L_{jh})$ and upper triangular matrix $U = (U_{hk})$ defined as follows:

$$L_{jh} = \binom{j-1}{h-1} \quad \text{and} \quad U_{hk} = \binom{t_0+1}{k-h}.$$

The equality $A = LU$ is a direct consequence of the "Vandermonde convolution" formula [10, eq. (5.22)]

$$\sum_{h=1}^{n} \binom{j-1}{h-1} \binom{t_0+1}{k-h} = \binom{t_0+j}{k-1}.$$

Both L and U have unit diagonals and are hence nonsingular: indeed, their inverses have the following simple explicit forms, as can be verified by direct multiplication using standard identities involving sums of products of binomial coefficients [10, eqs (5.24), (5.25)]:

$$(L^{-1})_{hk} = (-1)^{h+k} \binom{h-1}{k-1}$$

and

$$(U^{-1})_{jh} = (-1)^{j+h} \binom{t_0 + h - j}{t_0 + 1}.$$

Since $A^{-1} = U^{-1} L^{-1}$, the values $\{|P_k| : 1 \le k \le n\}$ may be computed in polynomial time using two matrix multiplications involving integers of $O(n \log n)$ bits. Observe that $\frac{1}{2}|P_1|$ gives the number of (unoriented) Hamilton paths in G.

The hardness of #HC is now simple to verify. Given a graph $G = (V, E)$ with the minimum degree condition we add a new vertex x and edges (x, v) for all $v \in V$ to create G'. Note that G' satisfies the minimum degree condition as well. Removing x from a Hamilton cycle in G' creates a Hamilton path in G. This defines a bijection from the set of Hamilton cycles in G' to the set of Hamilton paths in G. □

5. Counting the number of cycles of all sizes. We will first consider approximating the total number of cycles in graphs with minimum degree $(\frac{1}{2} + \alpha)n$.

We first note that if we add a loop to each vertex and extend the definition of 2-factor to include loops as cycles of length one, then the argument of [14] may be

extended to this case (note that we still forbid cycles of length 2, i.e., double edges). Thus there exists both a fully polynomial randomized approximation scheme and a fully polynomial almost uniform sampler for the set of *partial* 2-factors in a dense graph. Let a partial 2-factor be *cyclic* if it consists of a single cycle of length at least three and a collection of loops. Clearly, the number of cyclic partial 2-factors is the same as the number of cycles.

The procedure for approximating the number of cycles of all sizes is as follows: we estimate first the number of partial 2-factors in G, and then the number of cyclic partial 2-factors by standard sampling methods as a proportion of the number of partial 2-factors. To produce an accurate estimate in polynomial time it is only necessary to show that the ratio of partial 2-factors to cyclic partial 2-factors is not too large. Let

$$\mathcal{F}_\ell = \{\text{partial 2-factors with } \ell \text{ loops}\} \quad \text{and} \quad f_\ell = |\mathcal{F}_\ell|.$$

For a given $F \in \mathcal{F}_\ell$ let $L = \{\text{loops of } F\}$, which we will now identify with the corresponding set of vertices. For $v \in L$ let d_v denote the number of neighbors of v in L and $D = \sum_{v\in L} d_v$.

If $v \in L$ then there are at least $2\alpha n - 2d_v$ ways of adding v to a cycle C of F by deleting an edge (a, b) of C and adding edges $(a, v), (v, b)$. Indeed, we go round each cycle C of F; if the successor b of a vertex a neighboring v is also a neighbor of v, then it forms an (a, b, v) triangle. The number of such triangles is at least $2\alpha n - 2d_v$.

So in total there are at least

$$\sum_{v\in L}(2\alpha n - 2d_v) = 2\ell\alpha n - 2D \tag{4}$$

$$\geq 2\ell(\alpha n - (\ell - 1)) \tag{5}$$

such augmentations.

Suppose first that $\ell \leq \ell_1 = \lfloor \alpha n/2 \rfloor$. Then (5) gives at least $\ell\alpha n$ augmentations of $F \in \mathcal{F}_\ell$ to an $F' \in \mathcal{F}_{\ell-1}$. Each $F' \in \mathcal{F}_{\ell-1}$ arises in at most n ways and so

$$\frac{f_{\ell-1}}{f_\ell} \geq \alpha\ell.$$

Putting $\ell_0 = \lceil 2/\alpha \rceil$ we see that

$$f_{\ell_1} + f_{\ell_1-1} + \cdots + f_{\ell_0+1} \leq f_{\ell_0} \leq f_{\ell_0} + f_{\ell_0-1} + \cdots + f_0. \tag{6}$$

Suppose next that $\ell > \ell_1$. Note first that since a graph with r vertices and s edges contains at least $r - s + 1$ distinct cycles, we see that L contains at least

$$\frac{D}{2} - \ell + 1 \tag{7}$$

distinct cycles.

Adding a cycle C contained in L to F and removing $|C|$ loops gives us a 2-factor in $\mathcal{F}_{\ell'}$ where $\ell' < \ell$. From (4) and (7) we see that there are at least[2]

$$\left(\frac{2\ell\alpha n - 2D}{4}\right)^+ + \left(\frac{D}{2} - \ell\right)^+ \geq \ell\left(\frac{\alpha n}{2} - 1\right) \tag{8}$$

$$\geq \frac{\ell\alpha n}{3} \tag{9}$$

[2] $x^+ = \max\{0, x\}$.

augmentations of either sort from F. Each $F' \in \mathcal{F}_{<\ell}$ arises in at most $n+n$ ways (accounting for both ways of reducing L), and so

$$\begin{aligned} f_\ell &\le \frac{6}{\alpha\ell}(f_{\ell-1} + f_{\ell-2} + \cdots + f_0) \\ &\le \theta(f_{\ell-1} + f_{\ell-2} + \cdots + f_0), \end{aligned}$$

where $\theta = 12/(\alpha^2 n)$, assuming $\ell > \ell_1$.

Thus

$$\frac{f_\ell + f_{\ell-1} + \cdots + f_0}{f_{\ell-1} + f_{\ell-2} + \cdots + f_0} \le 1 + \theta,$$

and so

$$f_\ell + f_{\ell-1} + \cdots + f_0 \le (1+\theta)^{\ell-\ell_1}\Sigma_1, \tag{10}$$

where $\Sigma_1 = f_{\ell_1} + f_{\ell_1-1} + \cdots + f_0$. We weaken (10) to

$$\begin{aligned} f_{\ell_1+k} &\le (1+\theta)^k \Sigma_1 \\ &\le e^{12\alpha^{-2}}\Sigma_1. \end{aligned} \tag{11}$$

It follows from (6) and (11) that

$$\frac{f_0 + f_1 + \cdots + f_n}{f_0 + f_1 + \cdots + f_{\ell_0}} \le 2 + 2ne^{12\alpha^{-2}}. \tag{12}$$

Now take an $F \in \mathcal{F}_\ell$ where $\ell \le \ell_0$ and fix its set of loops L. The number of partial 2-factors with this same L is at most a polynomial factor, $p(n)$ say, of the number of cycles of size $n-\ell$ through $V \setminus L$, by the results of section 3. (It is clear that because ℓ is small here, the required degree conditions are satisfied.) Thus, by (12), the ratio of partial 2-factors to cyclic partial 2-factors is $O(np(n))$ and we have proved the existence of an fpras for the number of cycles.

6. Paths and Hamilton paths. We obtain an fpras for counting the number of Hamilton paths in the following way. We add a vertex v_0 and join it by an edge to every vertex of G. Call this new graph G^*. The number of Hamilton cycles in G^* is equal to the number of Hamilton paths in G. Since G^* is dense we can approximate the latter quantity by approximating the former.

Similarly, to estimate the number of paths of all lengths, we compute an estimate c^* for the number of cycles in G^* and an estimate ρ^* for the proportion ρ of cycles which contain v_0. Since the number of cycles containing v_0 is the number of paths in G, this provides an estimate $\rho^* c^*$ for the number of paths. Also, this will give us an fpras provided ρ is not too small. But clearly, $\rho \ge 3/4$ and we are done.

7. Concluding remarks. We remark that it is not difficult to adapt the above methods to the corresponding *directed* case. Here we will have both minimum indegree and outdegree at each vertex guaranteed to be at least $(\frac{1}{2}+\alpha)n$. Also, we may similarly count the number of connected k-factors in G for any $k = o(n)$. (Hamilton cycles are, of course, connected 2-factors.)

We leave open the following questions. First, is it possible to count approximately as $\alpha \to 0$ in any fashion? Second, is there a random walk on Hamilton cycles and (in some sense) "near-Hamilton cycles" which is rapidly mixing? In other words, can we avoid the Tutte construction and the need for 2-factors with many cycles?

Finally, are there other interesting counting problems which are tractable on such dense graphs? Note that Annan [1] has recently found an fpras for the number of spanning forests in a dense graph. This can easily be modified to approximate the total number of (not necessarily spanning) trees in a dense graph. On the other hand, Jerrum [11] has recently shown that the problem of computing this number for a general graph is #P-complete.

Acknowledgment. We thank Alistair Sinclair for his comments on an earlier draft.

REFERENCES

[1] J. D. Annan, *A randomised approximation algorithm for counting the number of forests in dense graphs*, Combin. Probab. Comput., 3 (1994), pp. 273–284.
[2] B. Bollobás, *Extremal Graph Theory*, Academic Press, Boston, 1978.
[3] G. R. Brightwell and P. M. Winkler, *Counting linear extensions*, Order, 8 (1991), pp. 225–242.
[4] A. Z. Broder, *How hard is it to marry at random?* (*On the approximation of the permanent*), in Proc. 18th ACM Symposium on Theory of Computing, 1986, pp. 50–58. Erratum in Proceedings of the 20th ACM Symposium on Theory of Computing, 1988, p. 551.
[5] G. A. Dirac, *Some theorems on abstract graphs*, in Proc. London Math. Soc., 2 (1952), pp. 69–81.
[6] M. E. Dyer and A. M. Frieze, *On the complexity of computing the volume of a polyhedron*, SIAM J. Comput., 17 (1988), pp. 967–974.
[7] M. E. Dyer, A. M. Frieze, and R. Kannan, *A random polynomial time algorithm for approximating the volume of convex bodies*, J. ACM, 38 (1991), pp. 1–17.
[8] A. M. Frieze, M. R. Jerrum, M. Molloy, R. Robinson, and N. Wormald, *Generating and Counting Hamilton Cycles in Random Regular Graphs*, Report ECS-LFCS-94-313, Department of Computer Science, University of Edinburgh, Scotland, December 1994.
[9] A. M. Frieze and S. Suen, *Counting Hamilton cycles in random directed graphs*, Random Structures Algorithms, 3 (1992), pp. 235–242.
[10] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, Addison-Wesley, Reading MA, 1989.
[11] M. R. Jerrum, *Counting trees in a graph is #P-Complete*, Inform. Process. Lett., 51 (1994), pp. 111–116.
[12] M. R. Jerrum and A. J. Sinclair, *Approximating the permanent*, SIAM J. Comput., 18 (1989), pp. 1149–1178.
[13] M. R. Jerrum and A. J. Sinclair, *Polynomial-time approximation algorithms for the Ising model*, SIAM J. Comput., 22 (1993), pp. 1087–1116.
[14] M. Jerrum and A. Sinclair, *Fast uniform generation of regular graphs*, Theoret. Comput. Sci., 73 (1990), pp. 91–100.
[15] R. M. Karp and M. Luby, *Monte-Carlo algorithms for enumeration and reliability problems*, in Proc. 24th IEEE Symposium on Foundations of Computer Science, 1983, pp. 56–64.
[16] A. Karzanov and L. G. Khachiyan, *On the Conductance of Order Markov Chains*, Technical Report DCS TR 268, Rutgers University, New Brunswick, NJ, 1990.
[17] M. Mihail and P. Winkler, *On the number of Euler orientations of a graph*, in Proc. 3rd Annual ACM-SIAM Symposium on Discrete Algorithms, 1992, pp. 138–145.
[18] J. Simon, *On the difference between one and many*, in Proc. 4th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science 52, Springer-Verlag, Berlin, 1977, pp. 480–491.
[19] W. T. Tutte, *A short proof of the factor theorem for finite graphs*, Canad. J. Math., 6 (1954), pp. 347–352.
[20] L. G. Valiant, *The complexity of computing the permanent*, Theoret. Comput. Sci., 8 (1979), pp. 189–201.
[21] L. G. Valiant, *The complexity of enumeration and reliability problems*, SIAM J. Comput., 8 (1979), pp. 410–421.

SIAM J. COMPUT.
Vol. 27, No. 5, pp. 1273–1282, October 1998
© 1998 Society for Industrial and Applied Mathematics
004

A SUBLINEAR SPACE, POLYNOMIAL TIME ALGORITHM FOR DIRECTED *s-t* CONNECTIVITY*

GREG BARNES†, JONATHAN F. BUSS‡, WALTER L. RUZZO§, AND BARUCH SCHIEBER¶

Abstract. Directed *s-t* connectivity is the problem of detecting whether there is a path from vertex s to vertex t in a directed graph. We present the first known deterministic sublinear space, polynomial time algorithm for directed *s-t* connectivity. For n-vertex graphs, our algorithm can use as little as $n/2^{\Theta(\sqrt{\log n})}$ space while still running in polynomial time.

Key words. graph connectivity, *s-t* connectivity, graph reachability, time-space tradeoff, JAG, NNJAG, NL

AMS subject classifications. 05C40, 05C85, 68Q05, 68Q10, 68Q15, 68Q20, 68Q25

PII. S0097539793283151

1. Introduction.

The *s-t* connectivity problem, detecting whether there is a path from a distinguished vertex s to a distinguished vertex t in a directed graph, is a fundamental one, since it is the natural abstraction of many computational search processes, and a basic building block for more complex graph algorithms. In computational complexity theory, it has an additional significance: understanding its complexity is a key to understanding the relationship between deterministic and nondeterministic space-bounded complexity classes. In particular, the *s-t* connectivity problem for *directed* graphs (STCON) is the prototypical complete problem for nondeterministic logarithmic space [12]. Both STCON and the undirected version of the problem, USTCON, are DLOG-hard—any problem solvable deterministically in logarithmic space can be reduced to either problem [7, 12].

Establishing the deterministic space complexity of STCON would tell us a great deal about the relationship between deterministic and nondeterministic space-bounded complexity classes. For example, showing a deterministic logarithmic space algorithm for directed connectivity would prove that $\mathrm{DSPACE}(f(n)) = \mathrm{NSPACE}(f(n))$ for any constructible $f(n) = \Omega(\log(n))$ [12]. Unfortunately, this remains a difficult open problem. A fruitful intermediate step is to explore *time-space tradeoffs* for STCON, that is, the *simultaneous* time and space requirements of algorithms for directed connectivity. No nontrivial lower bounds are known for general models of computation (such as Turing machines) on either the space or the simultaneous space and time required to solve STCON, although Cook and Rackoff [5] and Tompa [13] have obtained lower bounds for restricted models. This paper presents new upper bounds for the problem.

*Received by the editors May 11, 1993; accepted for publication (in revised form) July 18, 1996; published electronically May 19, 1998.

http://www.siam.org/journals/sicomp/27-5/28315.html

†Computing and Communications, University of Washington, Box 354843, Seattle, WA 98195–4843 (gsbarnes@u.washington.edu). This research was supported by NSF grant CCR-9002891.

‡Department of Computer Science, University of Waterloo, Waterloo, ON, Canada N2L 3G1 (jfbuss@math.uwaterloo.ca). This research was supported in part by a grant from NSERC.

§Computer Science and Engineering, University of Washington, Box 352350, Seattle, WA 98195–2350 (ruzzo@cs.washington.edu). This research was supported by NSF grant CCR-9002891.

¶IBM Research Division, T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598 (sbar@watson.ibm.com).

The standard algorithms for connectivity, breadth- and depth-first search, run in optimal time $\Theta(m+n)$ and use $\Theta(n \log n)$ space. At the other extreme, Savitch's theorem [12] provides a small space ($\Theta(\log^2 n)$) algorithm that requires time exponential in its space bound (i.e., time $n^{\Theta(\log n)}$). Cook and Rackoff show an algorithm for their more restricted "JAG" model that is similar to, but more subtle than, Savitch's; it has essentially the same time and space performance.

Recent progress has been made on the time-space complexity of USTCON. Barnes and Ruzzo [3] show the first sublinear space, polynomial time algorithms for undirected connectivity. Nisan [8] shows that $O(\log^2 n)$ space and polynomial time suffice. Nisan, Szemerédi, and Wigderson [9] show the first USTCON algorithm that uses less space than Savitch's algorithm ($O(\log^{1.5} n)$ versus $\Theta(\log^2 n)$).

Prior to the present paper, there was no corresponding sublinear space, polynomial time algorithm known for STCON, and there was some evidence suggesting that none was possible. It has been conjectured [4] that no deterministic STCON algorithm can run in simultaneous polynomial time and polylogarithmic space. Tompa [13] shows that certain natural approaches to solving STCON admit no such solution. Indeed, he shows that for these approaches, performance degrades sharply with decreasing space. Space $o(n)$ implies superpolynomial time, and space $n^{1-\epsilon}$ for fixed $\epsilon > 0$ implies time $n^{\Omega(\log n)}$, essentially as slow as Savitch's algorithm.

The main result of our paper is a new deterministic algorithm for directed s-t connectivity that achieves polynomial time and sublinear space simultaneously. While not disproving the conjecture of [4], it shows that the behavior elicited from certain algorithms by Tompa is not intrinsic to the problem. Our algorithm can use as little as $n/2^{\Theta(\sqrt{\log n})}$ space while still running in polynomial time. As part of this algorithm, we present an algorithm that finds short paths in a directed graph in polynomial time and sublinear space. The *short paths problem* is a special case of STCON that retains many of the difficulties of the general problem and seems particularly central to designing small space algorithms for STCON. We are not aware of any previous algorithms that solve this problem in sublinear space and polynomial time. Interestingly, our algorithm for the short paths problem is a generalization of two well-known algorithms for STCON. In one extreme it reduces to a variant of the linear time breadth-first search algorithm, and in the other extreme it reduces to the $O(\log^2 n)$ space, superpolynomial time algorithm of Savitch.

Subsequent to the appearance of a preliminary version of this paper [1], Poon [11, Chap. 4] has shown an algorithm for the JAG model that is similar to this algorithm and achieves similar performance. In addition, Barnes and Edmonds [2] and Edmonds and Poon [6] have given improved lower bounds for STCON on the JAG model and the stronger Node-Named JAG (*NNJAG*) model of Poon [10]. Edmonds and Poon's lower bound is particularly strong, since it nearly matches the upper bounds in this paper when the machine uses space $n/2^{O(\sqrt{\log n})}$ or less, and thus suggests that the algorithm in this paper could provide an optimal time-space tradeoff for STCON.

Our algorithm to solve STCON in polynomial time and sublinear space is constructed from two algorithms with different time-space tradeoffs. The first performs a modified breadth-first search of the graph, while the second finds short paths. Alone, neither algorithm can solve STCON in simultaneous polynomial time and sublinear space. In the following two sections, we present the breadth-first search algorithm and the short paths algorithm. Section 4 shows how the two algorithms can be combined to yield the desired result. Section 5 presents some notes and concluding remarks.

For more information on graph connectivity, see the survey paper by Wigderson [15].

2. The breadth-first search tradeoff. Consider the tree constructed by a breadth-first search beginning at s. The tree can contain n vertices and thus requires $O(n \log n)$ space to store. Instead of constructing the entire tree, our modified breadth-first search generates a fraction of the tree.

Suppose we want our modified tree to contain at most n/λ vertices. We can do this by only storing (the vertices in) every λth level of the tree. Number the levels of the tree $0, 1, \ldots, n-1$, where a vertex v is on level l if the shortest path from s to v is of length l. Divide the levels into equivalence classes $C_0, C_1, \ldots, C_{\lambda-1}$ based on their number mod λ. Besides s, the algorithm stores only the vertices in one equivalence class C_j, where j is the smallest value for which C_j has no more than the average number of vertices, n/λ.

The algorithm constructs this partial tree one level at a time. It begins with level 0, which consists of s only, and generates levels j, $j+\lambda$, $j+2\lambda$, $\ldots$, $j+\lambda \cdot \lfloor n/\lambda \rfloor$. Given a set, S, of vertices, we can find all vertices within distance λ of S in time $n^{O(\lambda)}$ and space $O(\lambda \log n)$ by enumerating all possible paths of length at most λ and checking which paths exist in G. This can be used to generate the levels of the partial tree. Let V_i be the vertices in levels 0, j, $j+\lambda$, $\ldots$, $j+i\lambda$. Consider the set of vertices, U, that are within distance λ of a vertex in V_i. Clearly, U contains all the vertices in level $j+(i+1)\lambda$. However, U may also contain vertices in lower numbered levels. The vertices in level $j+(i+1)\lambda$ are those vertices in U that are not within distance $\lambda - 1$ of a vertex in V_i. Thus, to get V_{i+1} we add to V_i all vertices that are within distance λ but not $\lambda - 1$ of V_i.

Pseudocode for the algorithm appears in Figure 2.1. Note that to find an equivalence class with at most n/λ vertices, the algorithm just tries all classes in order, discarding a class if it generates too many vertices.

Referring to Figure 2.1, the algorithm's space bound is dominated by the number of vertices in S and S', and the space needed to test whether a vertex is within distance λ of a vertex in S. There are never more than $n/\lambda + 1$ vertices in S and S', so the algorithm uses $O((n \log n)/\lambda)$ space to store these vertices. The time bound is dominated by repeatedly testing whether a vertex is within distance λ of a vertex in S. This test is performed $O(n^3/\lambda)$ times—the innermost loop to find the vertices on the next level of the tree makes $O(n \cdot n/\lambda)$ such tests (testing for a path from the $O(n/\lambda)$ vertices in S to all other $O(n)$ vertices), and is executed $O(\lambda \cdot n/\lambda)$ times.

In summary, we have shown the following.

THEOREM 2.1. *For any n-vertex directed graph and any integer $\lambda, 1 \leq \lambda \leq n$, the breadth-first search algorithm presented above solves s-t connectivity in space $O((n \log n)/\lambda + S_{PATH}(\lambda, n))$ and time $O((n^3/\lambda) \cdot T_{PATH}(\lambda, n))$, where $S_{PATH}(\lambda, n)$ and $T_{PATH}(\lambda, n)$ denote the space and time bounds, respectively, of the algorithm used to test for a path of length at most λ between two vertices in an n-vertex graph.*

Note that we assume that testing for a path of length at most j, $j-1$, or $\lambda - 1$ will not take asymptotically more time or space than testing for a path of length at most λ. This is because the first three problems are easily reduced to the latter. To test for a path of length at most λ', $\lambda' < \lambda$, from some vertex in a set S to some vertex v, connect $\lambda - \lambda'$ new vertices, $v_1, v_2, \ldots, v_{\lambda-\lambda'}$, in a chain to v by adding the edges $(v, v_1), (v_1, v_2), (v_2, v_3), \ldots, (v_{\lambda-\lambda'-1}, v_{\lambda-\lambda'})$. There will be a path in the new graph from some vertex $u \in S$ to $v_{\lambda-\lambda'}$ of length at most λ if and only if there was a path in the original graph from u to v of length at most λ'.

```
Algorithm Bfs (integer: λ);
                                  {remember every λth level of the breadth-first search tree}
  for j = 0 to λ − 1 do begin          {first level to remember, apart from level 0}
    S = {s}.
    for all vertices, v do begin                    {Find vertices on the first level}
      if v within distance j of s and
          v not within distance j − 1 of s then
        if |S| > n/λ then try next j.   {Don't store more than n/λ vertices, + vertex s}
        else add v to S.
    end;
    for i = 1 to ⌊n/λ⌋ do begin
      S′ = ∅.
      for all vertices, v do begin                  {Find vertices on the next level. ⋆}
        if v within distance λ of some vertex in S and
            v not within distance λ − 1 of any vertex in S then
          if |S| + |S′| > n/λ then try next j.
          else add v to S′.
      end;
      S = S ∪ S′.
    end;
    if t within distance λ of a vertex in S then return (Connected);
    else return (Not Connected);
  end;
end Bfs.
```

FIG. 2.1. *Details of the breadth-first search algorithm.*

Using a straightforward enumeration of all paths, testing whether a vertex is within distance λ requires $n^{O(\lambda)}$ time and $O(\lambda \log n)$ space. This algorithm is not sufficient for our purposes. In particular, if λ is asymptotically greater than a constant, the algorithm uses superpolynomial time. If we restrict our input to graphs with bounded degree, there is a slight improvement. In a graph where the outdegree is bounded by d, the number of paths of length λ from a vertex is at most d^{λ}. For these graphs, λ can be $O(\log n)$ and the algorithm will run in polynomial time. Note that the overall algorithm still does not use sublinear space in this case, even though the subroutine for finding paths of length λ does.

The problem with this algorithm is its method of finding vertices within distance λ. Explicitly enumerating all paths is not very clever, and uses too much time. There is hope for improvement, though, since this method uses only $O(\lambda \log n)$ space, much less than the $O(\frac{n}{\lambda} \log n)$ used by the rest of the algorithm. Indeed, in the next section we give an algorithm that uses more space but runs much faster.

3. The short paths tradeoff. Consider the *bounded s-t connectivity problem* (*bounded* STCON).

DEFINITION 3.1. *For any real-valued function $f(n)$, the $f(n)$-*bounded *s-t* connectivity problem *is, given an n-vertex directed graph G and two distinguished vertices s and t, to determine whether there is a path in G from s to t of length less than or equal to $f(n)$.*

Solving bounded STCON for general $f(n)$ is at least as hard as solving STCON. We are interested in the *short paths problem*, informally defined as the bounded STCON problem where $f(n)$ is small. The short paths problem is a special case of STCON

that seems to retain many of the difficulties of the general problem. It is particularly interesting given the breadth-first search algorithm above, because a more efficient method of finding short paths would clearly lead to an improvement in that algorithm's time bound.

Our second tradeoff is an algorithm that solves the short paths problem for many $f(n)$ in sublinear space and polynomial time. As will become clear, we will eventually want $f(n) = 2^{\Theta(\sqrt{\log n})}$, but to simplify the following discussion, we begin with the more modest goal of finding a sublinear space, polynomial time algorithm for the short paths problem with $f(n) = \log^c n$, for some integer constant $c \geq 1$.

As noted before, we already have a sublinear space, polynomial time algorithm that searches to distance $\log n$ on bounded degree graphs; because there are a constant number of ways to leave each vertex, we can enumerate and test all paths of length $\log n$ in polynomial time. In a general graph, this approach will not work, because there can be up to $n-1$ possible edges from each vertex, and explicit enumeration can yield a superpolynomial number of paths of length $\log n$. We can avoid this problem by using a labeling scheme that limits the number of possible choices at each step of the path.

Suppose we divide the vertices into k sets, according to their vertex number mod k. Then, every path of length L ($L = f(n)$) can be mapped to an $(L+1)$-digit number in base k, where digit i has value j if and only if the ith vertex in the path is in set j. Conversely, each such number defines a set of possible paths of length L.

Given this mapping, our algorithm is straightforward: generate all possible $(L+1)$-digit k-ary numbers, and check for each number whether there is a path in the graph that matches it. For a given k-ary number, the algorithm uses approximately $2n/k$ space to test for the existence of a matching path in the graph, as follows. Suppose we are looking for a path from s to t and want to test the $(L+1)$-digit number $\langle s \bmod k, d_1, d_2, \ldots, d_{L-1}, t \bmod k\rangle$. We begin with a bit vector of size $\lceil n/k \rceil$, which corresponds to the vertex set d_1. Zero the vector, and then examine the outedges of s, marking any vertex v in set d_1 (by setting the corresponding bit in the vector) if we find an edge from s to v. When we are finished, the marked vertices in the vector are the vertices in d_1 that have a path from s that maps to the first two digits of the number. Using this vector, we can run a similar process to find the vertices in d_2 that have a path from s that maps to the first three digits of the number, and store them in a second vector of size $\lceil n/k \rceil$. In general, given a bit vector of length $\lceil n/k \rceil$ representing the vertices in d_i with a path from s that maps to the first $i+1$ digits of the number, we use the other vector to store the vertices in d_{i+1} with a path from s that maps to the first $i+2$ digits. Pseudocode for the algorithm appears in Figure 3.1. Notice that the algorithm as given does not solve the short paths problem, as it tests for the existence of a path from s to t of length *exactly* L, not *at most* L. Any such algorithm can easily be converted into an algorithm for the short paths problem by adding a self-loop to s. For simplicity, we omit this detail from our algorithms.

The algorithm uses space $O(n/k)$ to store the vectors, and $O(L \log k)$ to write down the path to be tested. Let D be the maximum number of edges from one set of vertices d_i to another set of vertices d_j (i and j can be the same). For all steps in each path, we do at most $O(n/k + D)$ work zeroing the vector and testing for edges from d_{i-1} to d_i. Since $D = O(n^2/k^2)$, the algorithm uses $O(k^L L \cdot n^2/k^2) = O(k^L n^3)$ time to test all L steps on each of the k^L paths.

Unfortunately, this does not reach our goal of polynomial time and sublinear space when $L = \log^c n$. With a distance as small as $\log n$, k^L is only polynomial if k

```
Algorithm SP (integer: k, L; vertex s, t);
                                {Test for a path of length L between s and t using space O(n/k)}
  Create V_0 and V_1, two ⌈n/k⌉ bit vectors.
  for all (L + 1)-digit numbers in base k,
      ⟨d_0 = s mod k, d_1, ..., d_{L-1}, d_L = t mod k⟩ do begin
    Set all bits in V_0 to zero, and mark s (set the corresponding bit to 1).
    for i = 1 to L do begin
      Set all bits in V_{i mod 2} to zero.
                                                {Find edges from d_{i-1} to d_i}
      for all u in d_{i-1} marked in V_{(i-1) mod 2} and all v in d_i do begin
        if (u, v) is an edge then
          mark v in V_{i mod 2}.
      end;
    end;
    if t is marked in V_{L mod 2} then return (Connected);
  end;
  return (Not Connected);
end SP.
```

FIG. 3.1. *Details of the short paths algorithm.*

is constant, and if k is constant, the algorithm does not use sublinear space. We *can* achieve polynomial time and sublinear space by reducing the distance the algorithm searches. For example, if $L = \log n / \log\log n$, k can be $\log^c n$ for any constant c, and the algorithm will run in $O(n/\log^c n)$ space and $O((\log n)^{c \log n / \log\log n} n^3) = O(n^{c+3})$ time.

The algorithm can be improved by invoking it recursively. Consider the loop in the algorithm that tests for edges between one set of vertices and the next. This loop, in effect, finds paths of length one from marked vertices in the first set to vertices in the second set. Instead of finding paths of length one, we can use the short paths algorithm to find paths of length L, yielding an algorithm that uses twice as much space, but finds paths of length L^2. In general, using $r \geq 1$ levels of recursion, the improved algorithm can find paths of length L^r using $O(r(n/k + L \log k))$ space. If we make a recursive call for every possible pair of vertices in $d_{i-1} \times d_i$, we get a time bound of $O((k^L L \cdot n^2/k^2)^r) = O(n^{2r+1} k^{rL})$, since $L^r = O(n)$. In Figure 3.2, we present the pseudocode for our recursive algorithm. This algorithm uses a further refinement to improve the time bound—one recursive call is used to find all vertices in d_i reachable from any reachable vertex in d_{i-1}.

Given the discussion above, the time used by the recursive algorithm is bounded by the following recurrence relation, where $T(j)$ is the time used by the algorithm with j levels of recursion. For an appropriately chosen constant c,

$$T(j) = \begin{cases} O(n^2/k^2) & \text{if } j = 0, \\ k^L L(T(j-1) + cn/k) & \text{if } j > 0. \end{cases}$$

In the base case, the algorithm does $O(n^2/k^2)$ work. At other levels, the algorithm makes $k^L L$ recursive calls to itself, as well as doing some auxiliary work, such as setting all vector entries to zero. Solving the recurrence relation for $j = r$ gives time $O((k^L L)^r \cdot n^2/k^2) = O(n^3 k^{rL})$.

In summary, we have shown the following.

```
Algorithm SPR (integer: k, L, r, d_s, d_t; vector V_s): vector;
                          {Return the vector of vertices in set d_t that are reachable by paths
                            of length L^r from vertices in set d_s that are marked in vector V_s}
  Create V_0, V_1, and V_t, three ⌈n/k⌉-bit vectors. Set all bits in V_t to zero.
  if r = 0 then                                                          {base case}
    for all u in d_s marked in V_s and all v in d_t do begin
      if (u, v) is an edge then
        mark v in V_t.
    end;
  else
    for all (L + 1)-digit numbers in base k,
        ⟨d_0 = d_s, d_1, ..., d_{L-1}, d_L = d_t⟩ do begin
      V_0 = V_s.
      for i = 1 to L do begin                                  {Find paths from d_{i-1} to d_i}
        V_{i mod 2} = SPR(k, L, r - 1, d_{i-1}, d_i, V_{(i-1) mod 2}).
      end;
      Set all bits in V_t that are set in V_{L mod 2}.
    end;
  return (V_t);
end SPR.
```

FIG. 3.2. *Details of the recursive short paths algorithm.*

THEOREM 3.2. *For arbitrary integers r, k, and L, such that $r \geq 1$, $L \geq 1$, $n \geq k \geq 1$, and $L^r \leq n$, the recursive short paths algorithm, presented above, can search to distance L^r in time $O(k^{rL}L^r \cdot n^2/k^2)$ $(= O(n^3 k^{rL}))$ and space $O(r(n/k + L \log k))$.*

We will close this section with a few notes on the algorithm. This recursive algorithm meets our goal of finding a sublinear space, polynomial time algorithm that detects paths of polylogarithmic length. For example, for $L = \log n / \log\log n$, $k = \log^r n$, and constant $r \geq 2$, the algorithm searches to distance $L^r = \omega(\log^{r-1} n)$ in time $O(n^3 k^{rL}) = O(n^{r^2+3})$ and space $O(rn/\log^r n)$. However, as mentioned in the introduction, this algorithm does not by itself give a polynomial time, sublinear space algorithm for STCON. The algorithm searches to distance L^r by testing k^{rL} numbers. If $L^r = n$, then k^{rL} is polynomial only if $k = O(1)$. But if $k = O(1)$, the algorithm does not use sublinear space.

The algorithm, which was designed to solve the short paths problem, actually solves bounded STCON, and is thus a general algorithm for s-t connectivity. In fact, given the appropriate parameters, the algorithm exhibits behavior and performance similar to the best-known previous algorithms for STCON. If we let $k = 1$, $L = n$, and $r = 1$, the algorithm is a somewhat inefficient variant of breadth-first search that uses $O(n)$ space and $O(n(n + m))$ time: the algorithm first finds all vertices at distance 1 from s, then distance 2, etc., until it has searched to distance n. At the other end of the time-space spectrum, Savitch's algorithm is just the special case of this algorithm where $k = n$, $L = 2$, and $r = \lceil \log n \rceil$—this is also the minimum space bound for the algorithm.

4. Combining the two algorithms. As an immediate consequence of the previous two sections, we have an algorithm for STCON using sublinear space and polynomial time: use the modified breadth-first search algorithm to find every $(\log^c n)$th level of the tree (for integer constant $c \geq 2$), with the recursive short paths algorithm

```
                                  {S is the set of vertices on previous tree levels. S' (initially
                                   the empty set) will be the set of vertices on the next level}
for i_1 = 0 to k - 1 do begin
   S_{i_1} = {all vertices whose vertex number mod k = i_1}.
   P = ∅.                          {P will be all vertices in S_{i_1} on the next tree level}
   for i_2 = 0 to k - 1 do begin
      S_{i_2} = {all vertices whose vertex number mod k = i_2}.
      Q = S ∩ S_{i_2}.             {Q is all vertices in S_{i_2} on previous tree levels}
      A = {all vertices in S_{i_1} within distance L^r of a vertex in Q}.
      B = {all vertices in S_{i_1} within distance L^r - 1 of a vertex in Q}.
      P = P ∪ (A - B).
   end;
   if |S| + |S' ∪ P| > n/L^r then try next j.
   else S' = S' ∪ P.
end;
```

FIG. 4.1. *Combining the two algorithms efficiently.*

(the version that checks for paths of length up to L^r) as a subroutine to find the paths between levels. With careful choices of the parameters k, L, and r, however, the algorithm can use even less space while still maintaining polynomial time.

In general, if we set λ in the breadth-first search algorithm to be L^r, the breadth-first search algorithm finds every (L^r)th level of the tree, and the short paths algorithm searches to distance L^r. Substituting the space bound for the short paths algorithm (see Theorem 3.2) for the term $S_{PATH}(\lambda, n)$ in the breadth-first search algorithm (see Theorem 2.1), we get a space bound for this algorithm of

$$O((n \log n)/L^r + r(n/k + L \log k)), \tag{4.1}$$

where the first term corresponds to the space used by the partial breadth-first tree, and the second to the space used to find short paths. Substituting the short paths time bound for the term $T_{PATH}(\lambda, n)$ in the breadth-first search time bound gives a time bound of

$$O((n^3/L^r) \cdot k^{rL} L^r \cdot n^2/k^2) = O(n^5 k^{rL-2}).$$

The above time bound applies when we call the short paths algorithm every time the breadth-first search algorithm needs to know whether one vertex is within distance L^r of another. The two algorithms can be combined more efficiently by noticing that the short paths algorithm can answer many short paths queries in one call; for any pair of sets, (Q, R), such that R is one of the k sets of vertices in the short paths algorithm and Q is a subset of one of the k sets, one call to the short paths algorithm can be used to find all vertices in R within distance L^r of a vertex in Q. Thus, the short paths algorithm only needs to be called $2k^2$ times to generate the next level of the tree, twice for each possible pair of the k sets in the short paths algorithm. Figure 4.1 gives the code that should be used in place of the loop in Figure 2.1 (marked with a $\star$) that finds vertices on the next level of the breadth-first search tree. Similar code should replace the earlier loop in Figure 2.1 that finds the vertices on the first level.

The improved version makes a total of $O(k^2 n/L^r)$ calls to the short paths algorithm, for a time bound of

$$O((k^2 n/L^r) \cdot k^{rL} L^r \cdot n^2/k^2) = O(n^3 k^{rL}). \tag{4.2}$$

We want to find the minimum amount of space the algorithm can use while still maintaining a polynomial running time. To maintain polynomial time, we must have, for some constant a,

$$k^{rL} = n^a. \tag{4.3}$$

For simplicity, we bound expression (4.1) from below by

$$\Omega(n/L^r + n/k). \tag{4.4}$$

(That is, we omit the $\log n$ factor in the first summand and the r factor in the second summand, and leave out the third summand altogether.) The minimum value of the bound (4.4) is reached when the denominators are equal. For any given k, the product rL is fixed; thus the quantity L^r reaches its maximum, and the bound reaches its minimum, when L is a constant. Substituting L^r for k in (4.3) and solving for r yields $r = \sqrt{(a/L)\log_L n} = \Theta(\sqrt{\log n})$, and thus $k = 2^{\Theta(\sqrt{\log n})}$.

Substituting these values, $\sqrt{\log n}$ for r, $2^{\Theta(\sqrt{\log n})}$ for k, and a constant for L, into the simplified space bound expression (4.4) gives a bound of $n/2^{\Theta(\sqrt{\log n})}$. Substituting these same values into the actual space bound expression (4.1) yields the same asymptotic space bound, $n/2^{\Theta(\sqrt{\log n})}$. Since this matches the minimum for the simplified expression, which was a lower bound for this expression, we cannot do any better, and this must be the minimum space bound for the algorithm when using polynomial time.

The results of this section are summarized in the following theorem and its corollary.

THEOREM 4.1. *The combined algorithm, described above, solves* STCON *in space* $O((n \log n)/L^r + r(n/k + L \log k))$ *and time* $O(n^3 k^{rL})$, *for any integers* r, k, *and* L *that satisfy* $n \geq k \geq 1, r \geq 1, L \geq 1$, *and* $L^r \leq n$.

COROLLARY 4.2. *The combined algorithm can solve* STCON *in time* $n^{O(1)}$ *and space* $n/2^{\Theta(\sqrt{\log n})}$.

Proof. Choose $r = \sqrt{\log n}$, $k = 2^{\Theta(\sqrt{\log n})}$, and $L = 2$ in Theorem 4.1. As discussed above, these choices minimize space while retaining polynomial time. □

5. Conclusions and future work. Letting $L = 2$ and $k = 2^r$, we obtain the following corollary.

COROLLARY 5.1. *The combined algorithm of section* 4 *can solve* STCON *using time* $2^{O(\log^2(n/S))} \cdot n^3$ *given space* S.

The recent lower bound of Edmonds and Poon [6] shows that no algorithm that runs on an NNJAG can do better than time $2^{\Omega((\log^2(n/S))/\log\log n)}$ given space S. Ignoring the $\log\log n$ factor in the lower bound, the two bounds therefore match when $S = n/2^{O(\sqrt{\log n})}$. For higher space bounds, Edmonds and Poon's bound does not improve the JAG lower bound in Barnes and Edmonds [2] of $ST = \Omega(n^2/\log(n/S))$ (where T is the time used by the JAG).

Note that only the $\log\log n$ factor separates the lower bound from the upper bound when small space is used. When the space used is larger, there is still a

significant gap between the upper and lower bounds (although the gap is not as large as Theorem 4.1 indicates, since the time bound in (4.2) is an overestimate). It is possible that the combined algorithm above is optimal for the NNJAG model, but to prove it, these gaps must be eliminated.

Given that the upper and lower bounds for the problem are now close for the JAG and NNJAG models, we should consider more general models of computation. To improve this algorithm, it seems likely that we must use methods for exploring a graph that cannot be mimicked by an NNJAG. On the other hand, finding a lower bound similar to Barnes and Edmonds's [2] or Edmonds and Poon's [6] on a more general model of computation would be a breakthrough and would help decide the question of the complexity of *DL* versus *NL*.

Acknowledgments. Allan Borodin pointed us toward the short paths problem. Uri Feige helped find the minimum space bound.

REFERENCES

[1] G. Barnes, J. F. Buss, W. L. Ruzzo, and B. Schieber, *A sublinear space, polynomial time algorithm for directed s-t connectivity*, in Proc. 7th Annual IEEE Conference on Structure in Complexity Theory, Boston, MA, 1992, pp. 27–33.

[2] G. Barnes and J. A. Edmonds, *Time-space lower bounds for directed s-t connectivity on JAG models*, in Proc. 34th Annual IEEE Symposium on Foundations of Computer Science, Palo Alto, CA, 1993, pp. 228–237.

[3] G. Barnes and W. L. Ruzzo, *Undirected s-t connectivity in polynomial time and sublinear space*, Comput. Complexity, 6 (1996–1997), pp. 1–28.

[4] S. A. Cook, *Deterministic CFL's are accepted simultaneously in polynomial time and log squared space*, in Conference Record of the 11th Annual ACM Symposium on Theory of Computing, Atlanta, GA, 1979, pp. 338–345. See also [14].

[5] S. A. Cook and C. W. Rackoff, *Space lower bounds for maze threadability on restricted machines*, SIAM J. Comput., 9 (1980), pp. 636–652.

[6] J. A. Edmonds and C. K. Poon, *A nearly optimal time-space lower bound for directed st-connectivity on the NNJAG model*, in Proc. 27th Annual ACM Symposium on Theory of Computing, Las Vegas, NV, 1995, pp. 147–156.

[7] H. R. Lewis and C. H. Papadimitriou, *Symmetric space-bounded computation*, Theoret. Comput. Sci., 19 (1982), pp. 161–187.

[8] N. Nisan, $RL \subseteq SC$, Comput. Complexity, 4 (1994), pp. 1–11.

[9] N. Nisan, E. Szemerédi, and A. Wigderson, *Undirected connectivity in* $O(\log^{1.5} n)$ *space*, in Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science, Pittsburgh, PA, 1992, pp. 24–29.

[10] C. K. Poon, *Space bounds for graph connectivity problems on node-named JAGs and node-oriented JAGs*, in Proc. 34th Annual IEEE Symposium on Foundations of Computer Science, Palo Alto, CA, 1993, pp. 218–227.

[11] C. K. Poon, *On the Complexity of the st-Connectivity Problem*, Ph.D. Thesis, University of Toronto, 1996.

[12] W. J. Savitch, *Relationships between nondeterministic and deterministic tape complexities*, J. Comput. System Sci., 4 (1970), pp. 177–192.

[13] M. Tompa, *Two familiar transitive closure algorithms which admit no polynomial time, sublinear space implementations*, SIAM J. Comput., 11 (1982), pp. 130–137.

[14] B. von Braunmühl, S. A. Cook, K. Mehlhorn, and R. Verbeek, *The recognition of deterministic CFL's in small time and space*, Inform. and Control, 56 (1983), pp. 34–51.

[15] A. Wigderson, *The complexity of graph connectivity*, in Proc. 17th Symposium on Mathematical Foundations of Computer Science, I. M. Havel and V. Koubek, eds., Lecture Notes in Comput. Sci. 629, Springer-Verlag, New York, 1992, pp. 112–132.

SIAM J. COMPUT.
Vol. 27, No. 5, pp. 1283–1294, October 1998
© 1998 Society for Industrial and Applied Mathematics
005

MONOTONE CIRCUITS FOR CONNECTIVITY HAVE DEPTH $(\log n)^{2-o(1)}$*

MIKAEL GOLDMANN† AND JOHAN HÅSTAD‡

Abstract. We prove that a monotone circuit of size n^d recognizing connectivity must have depth $\Omega((\log n)^2/\log d)$. For formulas this implies depth $\Omega((\log n)^2/\log\log n)$. For polynomial-size circuits the bound becomes $\Omega((\log n)^2)$ which is optimal up to a constant.

Key words. monotone circuits, circuit complexity, connectivity, lower bounds

AMS subject classification. 68Q15

PII. S0097539795285631

1. Introduction.

1.1. Graph connectivity. Connectivity is the problem of determining if an undirected graph G is connected or not. This is a natural and fundamental problem which has been studied in numerous contexts. We refer the reader to Wigderson's survey of connectivity and its importance in complexity theory [9]. We consider the complexity of computing connectivity using monotone circuits with AND-gates and OR-gates of fan-in two. Typically, an n-node graph is encoded by $\binom{n}{2}$ variables $x_{i,j}$ that indicate whether $\{i,j\}$ is an edge in G or not. In [4] Karchmer and Wigderson proved that monotone circuits for the related problem (s,t)-connectivity (determining if there is an s,t-path in G) must have depth $\Omega((\log n)^2)$.[1] This is well known to be optimal. In spite of the two problems being very similar in nature, attempts to prove nontrivial bounds for connectivity were fruitless until Yao recently proved a lower bound of $\Omega((\log n)^{3/2}/\log\log n)$ [10].

While Karchmer and Wigderson used a top-down approach exploiting an equivalence between circuit depth and communication complexity (see [3]), Yao uses a bottom-up approach. In this article we modify his method to prove a lower bound of $\Omega((\log n)^2/\log\log n)$, and in the case of polynomial-size circuits the bound improves to optimal $\Omega((\log n)^2)$.

1.2. The method of approximation. The tool used by Yao in [10] is a modification of the method of approximation, originally designed by Razborov [7, 6] to prove lower bounds on the size of monotone circuits (also used in [1, 2]). The method is roughly as follows. One considers some subset of the inputs, called *test inputs*. Given some monotone circuit C one replaces each gate g by an approximator $\tilde{g}$ yielding a function $\tilde{C}$ that approximates the function that C computes. In order to prove a lower bound on the size of C one needs to show two things:

*Received by the editors May 8, 1995; accepted for publication (in revised form) July 18, 1996; published electronically May 19, 1998.

http://www.siam.org/journals/sicomp/27-5/28563.html

†Department of Numerical Analysis and Computing Science, KTH-NADA, 100 44 Stockholm, Sweden (migo@nada.kth.se). Research funded by post-doctoral fellowship from Swedish Research Council for Engineering Sciences. Research done as postdoc at the Laboratory for Computer Science, MIT.

‡Department of Numerical Analysis and Computing Science, KTH-NADA, 100 44 Stockholm, Sweden (ohanh@nada.kth.se). Research done as postdoc at the Laboratory for Computer Science, MIT. Research done while visiting Laboratory for Computer Science, MIT.

[1] All logarithms in this paper are to the base 2 unless we explicitly state otherwise.

1. g and $\tilde{g}$ agree on all but a tiny fraction of the test inputs,
2. C and $\tilde{C}$ disagree on a large fraction of the test inputs.

Since local errors are small but the total error is large, there must be many local errors; that is, C must have many gates.

Yao adapted the method of approximation to prove lower bounds on circuit depth. The key is to allow more and more powerful functions as approximators as one goes up the circuit. This way one can get good approximating functions at each level of the circuit. On the other hand, if there are not too many levels in the circuit, then the approximator one gets is still not powerful enough to agree with connectivity on most inputs.

Remark. The idea of having more powerful approximators as one goes up the circuit is also present in [5] and [8]. These papers prove lower bounds on the size of nonmonotone constant-depth circuits.

1.3. An overview of the paper. Recall the standard way to compute connectivity with $O(\log^2 n)$-depth monotone circuits. Let A_G be an $n \times n$ adjacency matrix of the undirected graph G and let I be the $n \times n$ identity matrix. Then G is connected if and only if $(A_G + I)^n$ (computed over the Boolean semiring $\langle \{0,1\}, \wedge, \vee \rangle$) is the all-one matrix. It is sufficient to look at $(A_G + I)^{2^{\lceil \log n \rceil}}$ which can be computed by $\lceil \log n \rceil$ squarings. Each squaring can be carried out by a $O(\log n)$-depth monotone circuit. If we look at an intermediate result, say $(A_G + I)^j$, then the u, v entry in this matrix tells us whether u and v are within distance 2^j in G or not. The intuition behind the proof is that this is more or less what a shallow monotone circuit can know about G: that some "small" sets of vertices are "close" to each other in G.

The basic approximators used have a parameter j that is coupled to the depth of the circuit that is to be approximated. A basic approximator checks if for some small subsets of the vertices all the vertices in each subset are within distance K^j in G (where K is a fixed parameter in the proof). For $j = 0$ the basic approximators are just checking edges in G, that is, looking at the inputs. Notice that the basic approximators correspond closely to the functions computed at the gates of the standard circuit above. The real approximators we use are disjunctions of basic approximators.

We wish to prove that some arbitrary shallow circuit C does not compute connectivity. Thus, we want to find an approximator that approximates C well. The proof proceeds in stages. At stage j we find approximators (with parameter j) for all the subfunctions computed by gates on level $j\epsilon \log n$ in C. As long as the circuit has depth $l\epsilon \log n$ where l is smaller than, say, $(\log_K n)/10$, the resulting approximator for the circuit only looks at local connectivity problems (checks if certain vertices are within distance $n^{1/10}$ in G) and therefore is poor at checking connectivity "globally."

In section 2 we present the approximators and prove some preliminary results about how they behave on certain "test inputs." We also prove some preliminary combinatorial lemmas.

In section 3 it is shown that monotone circuits of size n^d (where $d = o(\log n)$) for connectivity must have depth $\Omega((\log n)^2/\log d)$. By modifying the proof of this result we show in section 4 that monotone formulas for connectivity must have depth $\Omega((\log n)^2/\log\log n)$.

Finally, section 5 discusses the most obvious open problem: to show that computing connectivity with monotone formulas requires depth $\Omega((\log n)^2)$. We give some reasons why minor modifications to the method used in this paper probably will not be sufficient to prove this. In particular it is shown that there exists a circuit of depth $O((\log^2 n)/\log\log n)$ that when the method of approximation is applied to it, the

resulting function approximates connectivity well.

2. Test inputs and our approximating functions. As in Yao's paper we concentrate on two types of inputs:

1. Hamiltonian paths,
2. two disjoint cliques.

We use the abbreviation HP for the first type and 2C for the second. The goal is to show that an approximator for a shallow circuit either outputs 0 for a large fraction of HP or outputs 1 for a large fraction of 2C. We are interested in random instances of the two types and for HP we take the uniform distribution. A random instance of 2C is obtained by randomly and independently giving the labels 0 and 1 (each with probability 1/2) to each node. We then connect nodes with the same label. Thus, a random instance of HP is always connected while the probability that a random instance of 2C is connected is 2^{1-n} which for all practical purposes can be approximated with 0.

As mentioned in the introduction, we use approximations that are similar to Yao's. In other words, our method is bottom-up and starts with the inputs. We let the circuit do computation for $\epsilon \log n$ levels, and then we replace the functions computed by a nearby function. The information we concentrate on is the fact that certain subsets of vertices are known to be close to each other. The simplest case of this is an edge which is just saying that two vertices are at distance 1. To be able to formulate the general concept we need a little bit of notation.

DEFINITION 2.1. *Let $V = \{1, \dots, n\}$. A* partitioned subset *of V is a set $\mathbf{A} = \{A_1, \dots, A_k\}$ of disjoint subsets of V.*

Let $e(\mathbf{A}) = A_1 \cup \cdots \cup A_k$, and let the size of $\mathbf{A}$ *be $|e(A)|$.*

Let $s(\mathbf{A})$ be the number of sets in $\mathbf{A}$; i.e., $s(\mathbf{A}) = k$ in the given notation.

A partitioned set $\mathbf{A} = \{A_1, \dots, A_k\}$ is a subset *of another partitioned set $\mathbf{B} = \{B_1, \dots, B_l\}$ (written $\mathbf{A} \subseteq \mathbf{B}$) if $k \le l$ and (possibly after renumbering the parts of B) $A_i \subseteq B_i$ for $i = 1, 2, \dots, k$.*

A partitioned set $\mathbf{A}$ is finer *than a partitioned set $\mathbf{B}$ if for each $A_i \in \mathbf{A}$ there is a $B_j \in \mathbf{B}$ such that $A_i \subseteq B_j$. Conversely, we say that $\mathbf{B}$ is* coarser *than $\mathbf{A}$.*

It will be useful to have an estimate of the number of small partitioned subsets.

LEMMA 2.2. *Let $V = \{1, \dots, n\}$ and k be an integer. Then there are at most $(en)^k$ partitioned subsets of V of size at most k.*

Proof. We count by first picking a subset of V and then partitioning it. A set of size $s \le k$ can clearly be partitioned in at most $s^s \le k^k$ ways. Therefore, $k^k \sum_{s=0}^{k} \binom{n}{s}$ is an upper bound on the number of partitioned subsets of V. However,

$$\sum_{s=0}^{k} \binom{n}{s} \le \left(\frac{n}{k}\right)^k \sum_{s=0}^{n} \binom{n}{s} \left(\frac{k}{n}\right)^s = \left(\frac{n}{k}\right)^k \left(1 + \frac{k}{n}\right)^n \le \left(\frac{n}{k}\right)^k e^k,$$

which completes the proof. □

We will always have $|A_i| \ge 2$ and hence $|e(\mathbf{A})| \ge 2s(\mathbf{A})$.

The partitioned sets will play the role of minterms in our approximating functions. For a partitioned set $\mathbf{A}$ and integer r define the following function on graphs:

$$f^r_{\mathbf{A}}(G) = 1 \Leftrightarrow \bigwedge_{i=1}^{s(\mathbf{A})} \left(\bigwedge_{a,b \in A_i} d_G(a,b) \le r \right),$$

where $d_G(a,b)$ is the length of the shortest path in G between the nodes a and b. If a and b are not connected we define $d_G(a,b) = \infty$.

Our general approximating functions will be disjunctions of $f^r_{\mathbf{A}}$ for various $\mathbf{A}$ and r. Let $\mathcal{A} = \{\mathbf{A}^1, \ldots, \mathbf{A}^t\}$ be a collection of partitioned sets. Then we define

$$f^r_{\mathcal{A}} = \bigvee_{i=1}^{t} f^r_{\mathbf{A}^i}.$$

Also, for the empty set $\emptyset$ we define

$$f^r_{\emptyset} \equiv 1.$$

Boolean operations on functions $f^r_{\mathcal{A}}$ can almost be performed in the natural way. $f^r_{\mathcal{A}} \vee f^r_{\mathcal{B}}$ is of course $f^r_{\mathcal{A}\cup\mathcal{B}}$ so that is no problem. However, $f^r_{\mathcal{A}} \wedge f^r_{\mathcal{B}}$ is a little bit more complicated. By using the distributive law we need only to consider $f^r_{\mathbf{A}^i} \wedge f^r_{\mathbf{B}^j}$. If $e(\mathbf{A}^i)$ and $e(\mathbf{B}^j)$ are disjoint, this is just $f^r_{\mathbf{A}^i\cup\mathbf{B}^j}$, while if the two partitioned sets are not disjoint the resulting function might not be exactly representable and we can only find an approximation in our set of functions. Essentially we use $f^{r'}_{\mathbf{C}}$ where $e(\mathbf{C}) = e(\mathbf{A}^i) \cup e(\mathbf{B}^j)$ and the partition of $\mathbf{C}$ is the finest partition that is coarser than both $\mathbf{A}^i$ and $\mathbf{B}^j$ and r' is a suitable multiple of r. The fact that we are forced to increase r is one of the key reasons that we need to consider $\epsilon \log n$ levels at the time.

We start with an easy observation.

LEMMA 2.3. *$f^r_{\mathbf{A}}(G) = f^1_{\mathbf{A}}(G)$ for all $\mathbf{A}$ and all $r \geq 1$ and all graphs from $2C$.*

This is obvious since any two connected nodes are at distance 1.

In our arguments we want to keep our collections of partitioned sets small. One mechanism for doing this is to throw away any large partitioned set, and we need the following lemma.

LEMMA 2.4. *The probability that $f^r_{\mathbf{A}}(G) = 1$ for a random graph G from HP is at most*

$$\left(\frac{2r}{n - |e(\mathbf{A})|}\right)^{|e(\mathbf{A})|/2}.$$

Proof. Let $\mathbf{A} = \{A_1, \ldots, A_k\}$ and $a_i = |A_i|$. Define the notation $\mathbf{A}(i) = \{A_1, \ldots, A_i\}$ (this means that $\mathbf{A}(0) = \emptyset$). We have

$$\Pr[f^r_{\mathbf{A}}(G) = 1] = \prod_{i=1}^{k} \Pr[\, f^r_{\{A_i\}}(G) = 1 \mid f^r_{\mathbf{A}(i-1)}(G) = 1\,].$$

To estimate $\Pr[\, f^r_{\{A_i\}}(G) = 1 \mid f^r_{\mathbf{A}(i-1)}(G) = 1\,]$ let us assume that a random HP is found by randomly picking the places of all vertices one by one. We can pick the place for the first vertex in A_i in an arbitrary way, but for every other element of A_i there are at most $2r$ possible places which will satisfy the requirement. Since there are at least $n - e(\mathbf{A})$ remaining slots we get the bound

$$\begin{aligned}
\Pr[f^r_{\mathbf{A}}(G) = 1] &\leq \prod_{i=1}^{k} \left(\frac{2r}{n - |e(\mathbf{A})|}\right)^{a_i - 1} \\
&= \left(\frac{2r}{n - |e(\mathbf{A})|}\right)^{\sum_{i=1}^{k}(a_i - 1)} \\
&\leq \left(\frac{2r}{n - |e(\mathbf{A})|}\right)^{|e(\mathbf{A})|/2}.
\end{aligned}$$

The last inequality follows because $|e(\mathbf{A})| \geq 2k$. □

We will need a slightly more general statement of the lemma, and let us state this explicitly.

LEMMA 2.5. *Let $\mathbf{A}^i$, $i = 1, \ldots, d$, be a set of not necessarily disjoint partitioned sets and let $m = |\bigcup e(\mathbf{A}^i)|$. The probability that $\wedge_{i=1}^d f^r_{\mathbf{A}^i}(G) = 1$ for a random graph G from HP is at most*

$$\left(\frac{2r}{n-m}\right)^{m/2}.$$

Proof. The proof is almost identical to the proof of the previous lemma. Place the elements of $\bigcup e(\mathbf{A}^i)$ on the HP in random places. To satisfy that $\wedge_{i=1}^d f^r_{\mathbf{A}^i}(G) = 1$ at least half the elements will have at most $2r$ possible places to go. The lemma now follows. □

To avoid having too many large partitioned sets we will replace some collections by smaller collections and the concept of a sunflower is of central importance. Remember that the containment relation for partitioned sets respects the partition.

DEFINITION 2.6. *Let $\mathbf{A}^1, \ldots, \mathbf{A}^t$ be distinct partitioned sets. They form a t-*sunflower *with* core $\mathbf{C} = \{C_1, \ldots, C_k\}$ *if the following two conditions hold:*

1. $\mathbf{C} \subseteq \mathbf{A}^i$ *for* $1 \leq i \leq t$,
2. $e(\mathbf{A}^i) \cap e(\mathbf{A}^j) = e(\mathbf{C})$ *for* $1 \leq i < j \leq t$.

The petals *of a sunflower are the partitioned sets* $\mathbf{A}^i \setminus \mathbf{C} = \left\{ A \setminus e(\mathbf{C}) \mid A \in \mathbf{A}^i \right\}$.

Given a collection of partitioned sets that contain a sunflower with certain parameters, we will replace all the partitioned sets in the sunflower by the core. First note that this might create partitions with $|C_i| = 1$. These sets will simply be dropped when forming $f^r_{\mathbf{C}}$. We might also get an empty core and in such a case remember that $f^r_{\emptyset} \equiv 1$.

Replacing a t-sunflower by its core makes the corresponding function accept more inputs. We will not care that more HPs are accepted (they are actually quite a number). The reason is that we only worry about approximation errors that make the circuit accept less HP-inputs or more 2C-inputs. Thus, we need to check how many 2C-inputs get added by this procedure.

LEMMA 2.7. *Let $\mathbf{A}^1, \ldots, \mathbf{A}^t$ be partitioned sets which form a sunflower with core $\mathbf{C}$. Suppose that maximum of $|e(\mathbf{A}^i)| - |e(\mathbf{C})|$ is bounded by u; then the probability that a random element G of $2C$ satisfies $f^r_{\mathbf{C}}(G)$ while $f^r_{\mathbf{A}^i}(G) = 0$ for all i is bounded by $e^{-t2^{-u}}$.*

The process of replacing sunflowers by their core will be denoted *plucking*.

Proof of Lemma 2.7. We want to estimate the probability $\Pr[\vee_{i=1}^t f^1_{\mathbf{A}^i}(G) = 0 \mid f^1_{\mathbf{C}}(G) = 1]$. However, when G is drawn randomly under the condition that $f^1_{\mathbf{C}}(G) = 1$, then the events $f^1_{\mathbf{A}^i}(G) = 0$ are independent since $e(\mathbf{A}^i) \setminus e(\mathbf{C})$ is disjoint from $e(\mathbf{A}^j) \setminus e(\mathbf{C})$ when $i \neq j$.

$$\begin{aligned}
&\Pr[\vee_{i=1}^t f^1_{\mathbf{A}^i}(G) = 0 \mid f^1_{\mathbf{C}}(G) = 1] \\
&\quad = \prod_{i=1}^t \Pr[f^1_{\mathbf{A}^i}(G) = 0 \mid f^1_{\mathbf{C}}(G) = 1] \\
&\quad \leq (1 - 2^{-u})^t \\
&\quad \leq e^{-t2^{-u}}. \qquad \square
\end{aligned}$$

We need some information on the existence of sunflowers.

LEMMA 2.8. *We are given $\mathbf{A}^1, \ldots, \mathbf{A}^m$ that are distinct partitioned sets of size at most a, an integer t, and a partitioned set $\mathbf{B}$ of size b such that $\mathbf{B} \subseteq \mathbf{A}^i$ for $1 \leq i \leq m$. If $m \geq t^{a-b}(a-b)!a!/b!$, then there is a $\mathbf{C} \supseteq \mathbf{B}$ such that there is a t-sunflower with core $\mathbf{C}$.*

Proof. The proof is by induction on $a-b$. Since $\mathbf{A}^1, \ldots, \mathbf{A}^m$ are distinct we know that $a - b \geq 1$.

Base case ($a - b = 1$). The following greedy approach produces a t-sunflower. Pick an arbitrary partitioned set $\mathbf{A}^i$ and remove all sets $\mathbf{A}^j$ that intersect it outside $\mathbf{B}$. At first it might not seem like there can be any such $\mathbf{A}^j$. It is possible, however, that $e(\mathbf{A}^j) = e(\mathbf{A}^i)$, but the partitions are different. Since the partitions induced on $\mathbf{B}$ are the same, there may only be $\leq b = a - 1$ such sets. Since the total number of sets was at least ta we can repeat this $t - 1$ more times to get a t-sunflower with core $\mathbf{B}$.

Induction step ($a - b = i$).

Case 1. There is an element $x \notin \mathbf{B}$ appearing in at least $t^{a-b-1}(a-b-1)!a!/b!$ of the sets. There are at most $(b+1)$ ways to extend $\mathbf{B}$ by adding x to it, and there is at least one of these extensions, $\mathbf{B}'$, that is a subset of $t^{a-b-1}(a-b-1)!a!/(b+1)!$ of the $\mathbf{A}^i$. Since $a - |e(\mathbf{B}')| = i - 1$ one can use induction to find a t-sunflower among them with some core $\mathbf{C}$, where $\mathbf{B} \subseteq \mathbf{B}' \subseteq \mathbf{C}$.

Case 2. No element appears in more than $t^{a-b-1}(a-b-1)!a!/b!$ of the sets. Like in the base case we can pick a t-sunflower with $\mathbf{B}$ as its core in a greedy fashion. Each petal picked reduces the number of sets by at most $t^{a-b-1}(a-b)!a!/b!$ and thus we can pick t sets. □

Having established the basic preliminaries, let us repeat the outline of the proofs. We approximate the gates computed at levels $i\epsilon \log n$ by a function $f_{\mathcal{A}}^{K^i}$ where K will be a suitable number and $\mathcal{A}$ is a collection of partitioned sets of size $< K$ and which does not contain any sunflowers of certain parameters. For $i = 0$ there is no problem since the input $x_{i,j}$ is just $f_{\mathcal{A}}^1$ where $\mathcal{A}$ is just one partitioned set which contains the only set $\{i, j\}$. Let us now dive into the details and we start first with the case of when the circuits are of small size.

3. Polynomial size implies depth $\Omega((\log n)^2)$. The purpose of this section is to prove the following theorem.

THEOREM 3.1. *Given a monotone circuit of size n^d that computes connectivity, then the depth of this circuit is at least*

$$\frac{c(\log n)^2}{\log d}$$

for some universal constant c and sufficiently large n. Here d might be a function of n provided that it satisfies $d \in o((\log n)^{1/2})$.

In particular, the theorem implies that if the circuit is of polynomial size, then the depth is $\Omega((\log n)^2)$ and as is well known; this is tight.

To follow the general outline we just need to specify a couple of parameters. Set $\epsilon < 1/40$ such that $\epsilon \log n$ is an integer. The functions approximating the gates at level $i\epsilon \log n$ are functions $f_{\mathcal{A}}^{K^i}$, where

1. $K = 10d$;
2. $\mathcal{A}$ only contains partitioned sets of size at most K, and every sunflower has fewer than $2K2^K \log n$ petals.

Note that by Lemma 2.8 this implies that no collection of partitioned sets contains more than $(2K2^K \log n)^K (K!)^2 \leq n^\epsilon$ partitioned sets (for n sufficiently large).

Assume now that there is a circuit of size n^d and depth at most $\epsilon(\log n)^2/(4\log K)$ that computes connectivity. As described before, we will derive a contradiction by successively finding functions $f^r_{\mathcal{A}}$ that approximate the functions computed by gates in the circuit. The functions at the inputs ($i = 0$) are approximated perfectly, and the key is to go from i to $i+1$. Each gate at level $(i+1)\epsilon\log n$ is given by a circuit of depth $\epsilon\log n$ of gates at level $i\epsilon\log n$. There are at most n^d such gates and we approximate each gate separately.

Let us fix a gate at level $(i+1)\epsilon\log n$ and consider its $\epsilon\log n$ depth-defining circuit. The jth input is given by $f^{K^i}_{\mathcal{A}^j}$. We can convert the circuit to a depth-two circuit which is an $\vee$ of $\wedge$'s and such that the top fan-in is bounded by 2^{n^ϵ} and bottom fan-in is bounded by n^ϵ.

Since each $f^{K^i}_{\mathcal{A}^j}$ is conveniently represented as an $\vee$ of $\wedge$ we can just use the distributive law and compute each $\wedge$. This will produce a disjunction of functions g_α of the type $\wedge_{(j,k)\in\alpha} f^{K^i}_{\mathbf{A}^{j,k}}$ where the $\mathbf{A}^{j,k}$ are partitioned sets which might not be disjoint and α is a set of index pairs. We now proceed as follows.

1. Let $S_\alpha = \bigcup_{(j,k)\in\alpha} e(\mathbf{A}^{j,k})$.
2. Drop each g_α where $|S_\alpha| \geq K$.
3. Let $\mathbf{B}_\alpha$ be the finest partitioning of the set S_α that is coarser than $\mathbf{A}^{j,k}$ for all $(j,k) \in \alpha$.
4. Pluck the collection of $\mathbf{B}_\alpha$'s to form $f^{K^{i+1}}_{\mathcal{B}}$.

Let us look more closely at the approximations made. Dropping g_α with S_α large decreases the number of inputs that is accepted. We need to analyze the number of HPs dropped this way. This is done in Lemma 3.2.

When forming $\mathbf{B}_\alpha$ and increasing the value of allowable distances we accept more HPs. To see this, note that each set in $\mathbf{B}_\alpha$ is the union of at most $K-1$ sets $\mathbf{A}^{j,k}$. We must prove that any graph G that satisfies $\wedge_{(j,k)\in\alpha} f^{K^i}_{\mathbf{A}^{j,k}}(G) = 1$ also satisfies $f^{K^{i+1}}_{\mathbf{B}_\alpha}(G) = 1$. But for any pair $\{s,t\}$ of elements which are in the same set of $\mathbf{B}_\alpha$ there are elements $v_1 \ldots v_l$ with $l \leq K-1$ such that if we set $s = v_0$ and $t = v_{l+1}$ then for $r = 0, \ldots, l-1$ v_r and v_{r+1} are in the same set of $\mathbf{A}^{j,k}$ for some $(j,k) \in \alpha$. Since $\wedge_{(j,k)\in\alpha} f^{K^i}_{\mathbf{A}^{j,k}}(G) = 1$ we have $d_G(v_i, v_{i+1}) \leq K^i$ and hence $d_G(s,t) \leq K^{i+1}$. Since s and t were arbitrary, we conclude that $f^{K^{i+1}}_{\mathbf{B}_\alpha}(G) = 1$. By Lemma 2.3 we know that the subset of inputs from 2C that is accepted does not change when forming $\mathbf{B}_\alpha$.

Finally, plucking implies that we accept more inputs, and how many more inputs that are accepted from 2C is analyzed in Lemma 3.3.

We now prove the relevant lemmas.

LEMMA 3.2. *For sufficiently large n, the fraction of HP that satisfies any term dropped during step 2 of the construction is at most n^{-2d}.*

Proof. We find a small collection of functions h_β similar to g_α such that for any g_α dropped there is an h_β that covers g_α; that is, $g_\alpha(G) = 1 \Rightarrow h_\beta(G) = 1$ for some β.

For any dropped g_α find a minimal subset β of α such $|\bigcup_{(j,k)\in\beta} e(\mathbf{A}^{j,k})| \geq K$. Clearly there is such a β of size at most K. Let

$$h_\beta = \bigwedge_{(j,k)\in\beta} f^{K^i}_{\mathbf{A}^{j,k}}.$$

Then it clearly satisfies the property outlined above. Now we claim that

1. there are at most $2\binom{n^{2\epsilon}}{K}$ different h_β;

2. the probability that $h_\beta(G) = 1$ for a random HP G is at most $n^{-K/4}$ for sufficiently large n.

The first claim follows from the fact that each $\mathcal{A}^j$ contains at most n^ϵ partitioned sets and we need to choose at most K partitioned sets. The second claim follows from Lemma 2.5 (note that $r \le K^j \le K^{\log n/(4\log K)} \le n^{1/4}$). The lemma now follows since

$$2\binom{n^{2\epsilon}}{K} n^{-K/4} \le n^{-2d}$$

for $\epsilon < 1/40$ and sufficiently large n. □

Let us next estimate the number of inputs from 2C added under plucking.

LEMMA 3.3. *The fraction of* 2*C added in the above process is bounded by* $(e/n)^K$.

Proof. By Lemma 2.7, each time we replace a sunflower by its core we add a fraction n^{-2K} of 2C. This operation decreases the number of partitioned sets by at least one, and by Lemma 2.2 there are at most $(en)^K$ to begin with. The lemma now follows. □

Let us now finish the proof of the theorem. The output gate corresponds to $i = \log n/(4\log K)$ and hence it is approximated by a function $f_{\mathcal{A}}^{n^{1/4}}$. We have two cases

Case 1. We have the identically 0 function. In order for this to happen we must have lost all of HP. However, given that the circuit has size $\le n^d$ and using Lemma 3.2, the fraction of HP lost is at most $n^d \cdot n^{-2d} = n^{-d}$, which contradicts the assumption that all of HP has been lost.

Case 2. We get some function f which is not identically 0. There is a partitioned set $\mathbf{A}$ such that $|e(\mathbf{A})| \le K$ and $f_{\mathbf{A}}^{n^{1/4}}$ implies f. However, it is easy to see that the fraction of 2C accepted by $f_{\mathbf{A}}^{n^{1/4}}$, and thus by f is at least $2^{-K} = 2^{-10d}$. However, the total fraction of 2C added by the approximations is at most $n^d(e/n)^K = e^{10d}n^{-9d}$. For n sufficiently large, $e^{10d}n^{-9d} < 2^{-10d}$ and we have a contradiction.

4. Formulas require depth $\Omega((\log n)^2/\log\log n)$. The purpose of this section is to prove the following theorem.

THEOREM 4.1. *The depth of a monotone formula that computes connectivity is at least*

$$\Omega\left(\frac{(\log n)^2}{\log\log n}\right).$$

The outline is the same as in the other proof but we need to be more careful. In the sunflowers we remove, the number of petals we need is dependent on their sizes.

DEFINITION 4.2. *Let* $\mathbf{A}^1, \ldots, \mathbf{A}^t$ *be a sunflower with core* $\mathbf{C}$*, and say that the petal size* $|e(\mathbf{A}^i)| - |e(\mathbf{C})|$ *is at most* u*. We say that this is a* good *sunflower if* $t \ge 2^u(\log n)^2$.

Use the same ϵ as before. This time we use functions $f_{\mathcal{A}}^{K^i}$, where

1. $K = \lfloor \log n/(18\log\log n)\rfloor$,
2. $\mathcal{A}$ only contains partitioned sets of size at most K and no good sunflower.

Assume now that there is a circuit that has depth at most $\epsilon(\log n)^2/(4\log K)$ that computes connectivity. We proceed exactly as in the previous proof. In particular, the process of forming our approximations level by level is the same; i.e.,

1. let $S_\alpha = \bigcup_{(j,k)\in\alpha} e(\mathbf{A}^{j,k})$,

2. drop each g_α where $|S_\alpha| \geq K$,

3. let $\mathbf{B}_\alpha$ be the finest partition of the set S_α that is coarser than $\mathbf{A}^{j,k}$ for all $(j,k) \in \alpha$,

4. pluck the collection of $\mathbf{B}_\alpha$'s to form $f_{\mathcal{B}}^{K^{i+1}}$.

The crucial difference from the previous proof is that we need to work harder to estimate the number of HPs dropped at the first step. The crucial lemma follows.

LEMMA 4.3. *The fraction of HP dropped at a single gate is bounded by $n^{-K/8}$ for sufficiently large n.*

Proof. Again we form a small set of functions H that dominate the set of lost inputs.

Let $\gamma \subset \{1, \ldots, n^\epsilon\}$, and $|\gamma| = l \leq K$. Let $g_\gamma = \wedge_{j \in \gamma} f_{\mathcal{A}^j}^{K^i}$ and using the distributive law we can write

$$g_\gamma = \bigvee_{\delta} \bigwedge_{(j,k) \in \delta} f_{\mathbf{A}^{j,k}}^{K^i},$$

where δ ranges over all possible ways to pick $\mathbf{A}^{j,k}$ from $\mathcal{A}^j$ for each $j \in \gamma$. Call the term corresponding to δ h_δ and let H_γ be the set of terms in g_γ. We say that the *weight* of h_δ is $|\cup_{(j,k)\in\delta} e(\mathbf{A}^{j,k})|$. We call h_δ *heavy* if its weight is at least K and let H_γ^h be the set of heavy terms in H_γ. Finally, let $H = \cup_{|\gamma| \leq K} H_\gamma^h$. Now, H covers all sets dropped in step 2 for the following reason. For any α with $|S_\alpha| \geq K$ pick a subset β of size at most K such that $|\bigcup_{(j,k)\in\beta} e(\mathbf{A}^{j,k})| \geq K$. The term that corresponds to α is covered by the term $h_\beta = \wedge_{(j,k)\in\beta} f_{\mathbf{A}^{j,k}}^{K^i}$. Let γ be the projection of β on the first coordinate. Clearly, $h_\beta \in H_\gamma^h \subseteq H$.

It remains to analyze the fraction of HP that is accepted by any of the functions in H. For any $h \in H$ let s be its weight. By definition $s \geq K$, and since we only consider $|\gamma| \leq K$ we have $s \leq K^2$. First note that by Lemma 2.5 the fraction of HP that satisfies a term h_δ of weight s is at most $n^{-s/3}$ for n sufficiently large. Now we need the following lemma.

LEMMA 4.4. *Given γ, $|\gamma| = l \leq K$. The number of h in H_γ of weight s is at most*

$$F(s,l) = 2^{Ks + 2s\log\log n + 2s\log s + l(s\log s + s + 1)}.$$

Before we prove Lemma 4.4, let us just see how to complete the proof of Lemma 4.3. For a fixed γ we drop at most a fraction $\sum_{s=K}^{K^2} F(s,K) n^{-s/2}$ of HP. For n sufficiently large this quantity is bounded by

$$K^2 \cdot 2^{3K^2 \log\log n - (K/3)\log n} = K^2 \cdot n^{-K/6}.$$

There are fewer than $n^{\epsilon K}$ sets γ, and for n sufficiently large, we have that $K^2 \cdot n^{(\epsilon - 1/6)K} < n^{-K/8}$. □

Let us now prove Lemma 4.4.

Proof. We need the following useful technical lemma.

LEMMA 4.5. *We are given $\mathbf{A}^1, \ldots, \mathbf{A}^m$ that are distinct partitioned sets, a set R of size r such that $|R \cup e(\mathbf{A}^i)| \leq s$ for $1 \leq i \leq m$, and an integer t.*

If $m \geq (r+1)^r t^{s-r}(s-r)!s!/r!$, then there is a $\mathbf{C}$ such that there is a t-sunflower with core $\mathbf{C}$, and for each $\mathbf{A}^i$ in the sunflower, $|e(\mathbf{A}^i) \setminus e(\mathbf{C})| \leq s - r$.

Proof. We will find a subset of R and partition it to get a partitioned set $\mathbf{Q}$ such that for many of the sets $\mathbf{Q} \subseteq \mathbf{A}^i$ and $|e(\mathbf{A}^i) \setminus e(\mathbf{Q})| \leq s - r$, and then use Lemma 2.8.

Let $R^i = R \cap \mathbf{A}^i$, and let $\mathbf{R}^i$ be the partitioned set that $\mathbf{A}^i$ induces on R^i. Since $|R^i| \leq r$, $(r+1)^r$ is an upper bound on the number of distinct $\mathbf{R}^i$ that can be obtained. Therefore at least $m' = t^{s-r}(s-r)!s!/r!$ of the $\mathbf{A}^i$ must give the same partitioned set. Call this partitioned set $\mathbf{Q}$ and let $q = |e(\mathbf{Q})|$.

Without loss of generality $\mathbf{A}^1, \ldots, \mathbf{A}^{m'}$ all give $\mathbf{Q}$. Since for these, $\mathbf{Q} \subseteq \mathbf{A}^i$ and $|e(\mathbf{A}^i)| \leq s - r + q$, we can apply Lemma 2.8 with $a = s - r + q$, $b = q$, and $\mathbf{B} = \mathbf{Q}$. Provided that $m' \geq t^{s-r}(s-r)!(s-r+q)!/q!$ we are done. The right-hand side of this expression grows with q, so $m' \geq t^{s-r}(s-r)!s!/r!$ suffices, and we have completed the proof. □

Now we can establish Lemma 4.4 by induction on l. We want to show that the number of terms of size s of H_γ is bounded by $F(s,l)$. The base case $l = 1$ follows from Lemma 2.8 with $\mathbf{B}$ being the empty set. Obviously there are no terms of weight $s \geq K$. For weight $s < K$ we just need to observe that

$$(2^s(\log n)^2)^s s! \leq 2^{Ks+2s\log\log n + s\log s} \leq F(s,1).$$

For the induction step consider $\gamma = \{\gamma_1, \ldots, \gamma_l\}$, and let $\gamma' = \{\gamma_1, \ldots, \gamma_{l-1}\}$. To do the induction we need to analyze how many terms of size s in H_γ can be formed from a single term of size $r \leq s$ in $H_{\gamma'}$. Since $\mathcal{A}^{\gamma_l}$ does not contain any good sunflowers, by Lemma 4.5 with parameter $t = 2^{s-r}(\log n)^2$ we conclude that each term of size r in $H_{\gamma'}$ can give at most

$$\begin{aligned}&(r+1)^r(2^{s-r}(\log n)^2)^{s-r}(s-r)!s!/r! \\ &\quad \leq 2^{(s-r)^2+2(s-r)\log\log n+2(s-r)\log s+r\log r+r}\end{aligned}$$

different terms of size s in H_γ. The inequality is immediate except for $(s-r)!s!/r! \leq 2^{2(s-r)\log s}$, but this follows because the left-hand side has $2(s-r)$ factors that are all at most s. Also, note that a term of size r cannot give any terms of size $s > r + K$ since all $\mathbf{A} \in \mathcal{A}^{\gamma_l}$ have size at most K.

To get the total number of terms of size s we just use the inductive hypothesis to bound the number of terms of size r in $H_{\gamma'}$ and sum over $r \geq s - K$. Through simple calculation,

$$\begin{aligned}&\sum_{r=\max(0,s-K)}^{s} F(r,l-1)\cdot 2^{(s-r)^2+2(s-r)\log\log n+2(s-r)\log s+r\log r+r} \\ &\quad \leq F(s,l). \qquad \square\end{aligned}$$

The fraction of 2C lost by plucking is easy to estimate.

LEMMA 4.6. *The fraction of $2C$ inputs added at a single gate is bounded by $n^{-\log n}$ for sufficiently large n.*

Proof. By Lemma 2.7 and the definition of good sunflowers we know that plucking a single good sunflower adds a fraction at most $e^{-(\log n)^2}$ of 2C. Each time we pick a sunflower the number of partitioned sets decreases, and by Lemma 2.2 there are at most $(en)^K$ partitioned sets of size $\leq K$ before plucking starts. Thus, the fraction of 2C that gets added at a single gate is less than $e^{-(\log n)^2 + K\log n + O(K)}$, which is less than $n^{-\log n}$ for sufficiently large n. □

The proof of Theorem 4.1 is now completed in exactly the same way as Theorem 3.1 by using Lemmas 4.3 and 4.6 instead of Lemmas 3.2 and 3.3 and the fact that the size of the formula is at most $n^{\epsilon \log n/(4\log K)}$. For n sufficiently large there are less than $n^{\log n/(150\log\log n)}$ gates, and at each gate the error on HP is at most $n^{-K/8} \leq n^{-\log n/(145\log\log n)}$ and the error on 2C at each gate is at most $n^{-\log n}$.

5. Conclusion and open problems. Combining the results of the previous two sections shows that a monotone circuit of size n^d for connectivity must be of depth $\Omega((\log n)^2/\log d)$. Of course one would like to get $\Omega((\log n)^2)$ lower bounds for the depth even for formulas. We do believe that this is the correct answer, but we believe that it is hard to find a more or less straightforward extension of the current method. The following intuitive argument explains the problem.

An approximation argument seems to require partitioned sets[2] of size $\Omega(\log n)$. The reason is that if we drop a single partitioned set of size K then the fraction of HPs dropped is at least n^{-K}. Since we are discussing circuits of size $n^{c\log n}$ we cannot afford this if $K = o(\log n)$. Now suppose we are using some type of sunflowers and plucking. Consider a program that computes a predicate $D(s,t,i)$ recursively where s and t are vertices and i is a parameter. It sets $D(s,t,0)$ to true iff (s,t) is an edge. To compute $D(s,t,i)$ set $s = v_0$, $t = v_l$ and try n^2 ways of picking $v_1 \dots v_{l-1}$ and set $D(s,t,i)$ to true if for some attempt $D(v_i, v_{i+1}, i)$ are true for $i = 0, 1, \dots, i-1$. $D(s,t,i)$ should be thought of as a crude approximation that s and t are within distance l^i. This is not really true since we do not try all the values of the $l-1$ intermediate points. However, in an approximation scheme such as ours $D(s,t,i)$ is converted into the function $d_G(s,t) \le l^i$ by plucking (this argument is inductive on i). If we choose $l = \sqrt{\log n}$ then $D(s,t,i)$ can be computed in depth about $O(i \log n)$ and for $i = i_0 = 2\log n/\log\log n$ the approximation of $D(s,t,i)$ is whether s and t are connected and hence $\wedge_{t=2}^{n} D(1,t,i_0)$ is approximated by connectivity.

The problem arises because the plucking operation causes a significant increase in the fraction of HPs accepted. In our opinion, a major idea or a totally different approach seems to be needed to eliminate the $\log\log n$ factor.

Acknowledgment. The results described were obtained while visiting the Lab for Computer Science at MIT. We thank the LCS and in particular Mike Sipser for this opportunity. We are also grateful to the referees for careful reading and helpful suggestions.

REFERENCES

[1] N. Alon and R. B. Boppana, *The monotone circuit complexity of boolean functions*, Combinatorica, 7 (1987), pp. 1–22.

[2] A. E. Andreev, *On a method for obtaining lower bounds for the complexity of individual monotone functions*, Dokl. Akad. Nauk SSSR, 282 (1985), pp. 1033–1037 (in Russian); Soviet Math. Dokl., 31 (1985), pp. 530–534 (in English).

[3] M. Karchmer, *Communication Complexity: A New Approach to Circuit Depth*, The MIT Press, Cambridge, MA, 1989.

[4] M. Karchmer and A. Wigderson, *Monotone circuits for connectivity require super-logarithmic depth*, SIAM J. Discrete Math., 3 (1990), pp. 255–265.

[5] A. A. Razborov, *Lower bounds on the size of bounded-depth networks over a complete basis with logical addition*, Mat. Zametki, 41 (1987), pp. 598–607 (in Russian); Math. Notes of the Academy of Sciences of the USSR, 41 (1987), pp. 333–338 (in English).

[6] A. A. Razborov, *Lower bounds on the monotone complexity of some boolean functions*, Dokl. Akad. Nauk SSSR, 281 (1985), pp. 798–801 (in Russian); Soviet Math. Dokl., 31 (1985), pp. 354–357 (in English).

[7] A. A. Razborov, *A lower bound on the monotone network complexity of the logical permanent*, Mat. Zametki, 37 (1985), pp. 887–900 (in Russian); Math. Notes of the Academy of Sciences of the USSR, 37 (1985) pp. 485–493 (in English).

[2] Of course, we might use something different than partitioned sets, for instance the same graphs as Yao did. This, however, does not matter greatly for this argument.

[8] R. Smolensky, *Algebraic methods in the theory of lower bounds for boolean circuit complexity*, in Proc. 19th ACM Symposium on Theory of Computing, New York, 1987, pp. 77–82.

[9] A. Wigderson, *The complexity of graph connectivity*, in Mathematical Foundations of Computer Science, I. Havel and V. Koubek, eds., Lecture Notes in Computer Science 629, Springer-Verlag, Berlin, Heidelberg, 1992, pp. 112–132.

[10] A. C. Yao, *A lower bound for the monotone depth of connectivity*, in Proc. 35th IEEE Symposium on Foundations of Computer Science, Sante Fe, NM, 1994, pp. 302–308.

SIAM J. COMPUT.
Vol. 27, No. 5, pp. 1295–1302, October 1998
© 1998 Society for Industrial and Applied Mathematics
006

COMPLEXITY ANALYSIS OF A PARALLEL LATTICE BASIS REDUCTION ALGORITHM*

C. HECKLER† AND L. THIELE‡

Abstract. Lattice basis reduction is an important problem in geometry of numbers with applications in combinatorial optimization, computer algebra, and cryptography. The well-known sequential LLL algorithm finds a short vector in $O(n^4 \log B)$ arithmetic operations on integers having binary length $O(n \log B)$, where n denotes the dimension of the lattice and B denotes the maximum L_2 norm of the initial basis vectors. In this paper a new analysis of the parallel algorithm of Roch and Villard is presented. It is shown that on an $n \times n$ mesh it needs $O(n^2 \log B)$ arithmetic operations on integers having binary length $O(n \log B)$. This improves the previous analysis and shows that an asymptotical speedup of n^2 is possible using n^2 processors.

Key words. lattice basis reduction, parallel algorithms, parallel computational complexity, size of the numbers

AMS subject classifications. 11H06, 11Y16, 68Q22

PII. S0097539795295626

1. Introduction.

Lattice basis reduction, the problem of finding a basis of a lattice with "short" vectors, is an important problem in geometry of numbers with applications in combinatorial optimization, computer algebra, and cryptography (see, for example, [8, 7, 6]). In the field of basis reduction a major advance was reached in 1982 when A.K. Lenstra, H.W. Lenstra Jr., and L. Lovasz [7] developed the so-called LLL algorithm, which is a polynomial-time algorithm finding a "good" basis ("reduced basis") in $O(n^4 \log B)$ arithmetic operations on integers having binary length $O(n \log B)$, where n denotes the dimension of the lattice and B denotes the maximum L_2-norm of the initial basis vectors. Subsequently various variants have been developed; see, for example, [10, 11]. In 1992 Roch and Villard [9, 13] developed a parallel algorithm for n^2 processors, but the efficiency of that algorithm could not be proved. Moreover, larger numbers than in the sequential LLL algorithm can occur in this parallel algorithm. They show the bound $O(n^2 \log B)$ for the binary length of the integers. In this paper a new analysis of the parallel algorithm is presented. It is shown that on an $n \times n$ mesh it needs $O(n^2 \log B)$ arithmetic operations and communication steps on integers having binary length $O(n \log B)$. Thus, using n^2 processors an optimal speedup can be reached.

This paper is organized as follows: the next section gives a brief survey of the notation used in the paper. In section 3 the parallel algorithm is reviewed. The complexity in terms of the number of arithmetic operations is studied in section 4. Finally, in section 5 the size of the numbers of the serial and parallel algorithms and the binary complexity are considered.

*Received by the editors December 1, 1995; accepted for publication (in revised form) July 19, 1996; published electronically May 19, 1998.
http://www.siam.org/journals/sicomp/27-5/29562.html

†Universität-GH Paderborn, Fachbereich 17, Automath, Warburgerstr. 100, D-33098 Paderborn, Germany (chh@uni-paderborn.de). The research of this author was done at the Universität des Saarlandes, Lehrstuhl für Mikroelektronik, Postfach 151150, D-66041 Saarbrücken and was supported by the Deutsche Forschungsgemeinschaft, SFB 124.

‡ETH Zürich, Technische Informatik und Kommunikation, ETH-Zentrum, CH-8092 Zürich, Switzerland (thiele@tik.ee.ethz.ch). This research was done while this author was at the Universität des Saarlandes, Saarbrücken, Germany.

A more practical algorithm using fast floating point arithmetic can be found in [3]. Implementation details and practical results of different parallel algorithms are given in [2].

2. Basic notation. We assume that the reader is familiar with the foundations of lattice basis reduction as described in section 1 of [7] (also see, for example, [11, 12]). Here we only describe our basic notation.

An n-dimensional lattice is a discrete subgroup of $\mathbf{R}^n$.

DEFINITION 2.1. *Let $b_1, b_2, \ldots, b_n \in \mathbf{R}^n$ be n linearly independent vectors. Then $L = L(b_1, \ldots, b_n) = \{\sum_{i=1}^{n} \lambda_i b_i | \lambda_i \in \mathbf{Z}, i = 1, \ldots, n\}$ is called a lattice with basis $(b_1, \ldots, b_n)$.*

Let us recall the Gram–Schmidt orthogonalization process. Let $\| \cdot \|$ denote the L_2-norm of a vector and let $(b_1, b_2, \ldots, b_n)$ be a basis, then the Gram–Schmidt orthogonalization $(b_1^*, b_2^*, \ldots, b_n^*)$ of mutually orthogonal vectors can be computed by $b_1^* = b_1$ and $b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^*$ for $i = 2, \ldots, n$, where $\mu_{i,j} = \frac{b_i^T b_j^*}{||b_j^*||^2}$. Let be $\mu_{i,i} = 1$ and $\mu_{i,j} = 0$ for $i < j$, and let $M = (\mu_{i,j})_{1 \le i,j \le n}$ be the matrix of the Gram–Schmidt coefficients. Then M is a lower triangular matrix where all elements of the main diagonal are equal to 1 and the above equations can be expressed as $(b_1, \ldots, b_n) = (b_1^*, \ldots, b_n^*) M^T$.

The following procedure (designated "size reduction") is used to obtain a basis of the lattice with small Gram–Schmidt coefficients $|\mu_{i,j}| \le 0.5$.

ALGORITHM 2.2.

Size reduction of the basis $(b_1, \ldots, b_n)$ [7]

```
for i = 2 to n do
    reduce size of column i of the basis and of M^T:
      for j = i − 1 downto 1 do
          reduce size of coefficient μ_{i,j} of column i:
            let ⌈μ_{i,j}⌋ be the nearest integer to μ_{i,j}
            and let μ_j denote the column j of M^T:
            b_i ← b_i − ⌈μ_{i,j}⌋ b_j; μ_i ← μ_i − ⌈μ_{i,j}⌋ μ_j
      od
od
```

For technical reasons we use the following definition of a reduced basis (due to [12]) which is almost the same as the original one given in [7].

DEFINITION 2.3 (reduced basis [7, 12]). *A lattice basis $(b_1, \ldots, b_n)$ is called reduced if both of the following conditions are fulfilled:*

- $|\mu_{i,j}| \le \frac{1}{2}$ *for all* $1 \le j < i \le n$,
- $||b_{i-1}^*||^2 \le 2||b_i^*||^2$ *for all* $1 < i \le n$.

The idea of the LLL lattice basis reduction algorithm of [7] consists in iteratively computing the size reduction of one basis vector b_i and swapping b_i and b_{i-1} if the second condition of the above definition is not valid. The complexity of the LLL algorithm is given in the next theorem.

THEOREM 2.4 (sequential complexity [7]). *Let $B = \max_{i=1,\ldots,n} ||b_i||^2$ be the square of the length of the longest vector of the original basis. Given a basis $(b_1, \ldots, b_n)$ of a lattice with $b_i \in \mathbf{Z}^n$ for all $1 \le i \le n$, the sequential LLL algorithm of [7] computes a reduced basis of the same lattice in $O(n^4 \log B)$ arithmetic operations on integers having binary length $O(n \log B)$.*

3. The parallel algorithm. The main idea of the parallel algorithm of [9, 13] is to perform all possible swaps in one phase (see also [3, 4]) and to compute the size

reduction of the whole matrix. Strictly speaking, b_i and b_{i+1} are swapped for *all odd* indices i with $||b_i^*||^2 > 2||b_{i+1}^*||^2$ in one phase of the algorithm and for *all even* indices i in the next phase of the algorithm (see Algorithm 3.1). The concatenation of an odd and an even phase is called an *"all swap phase."*

ALGORITHM 3.1.

All-Swap Lattice Basis Reduction Algorithm [3, 4, 9, 13]
input: $b_1, b_2, \ldots, b_n$ a lattice basis
output: $b_1, b_2, \ldots, b_n$ a reduced basis
method:

```
Compute the Gram–Schmidt orthogonalization of
the basis, i.e. M and ‖b_i^*‖^2, i = 1, ..., n;
ordering ← odd;
while (any swap is possible) do
    Compute the size reduction of the basis (b_1, ..., b_n) and
    the Gram–Schmidt coefficients M;
    if (ordering = odd) then
        swap b_i and b_{i+1} for every odd index i with
        ||b_i^*||^2 > 2||b_{i+1}^*||^2;
        ordering ← even;
    else
        swap b_i and b_{i+1} for every even index i with
        ||b_i^*||^2 > 2||b_{i+1}^*||^2;
        ordering ← odd;
    fi;
    Repair the Gram–Schmidt orthogonalization
od.
```

In the serial case, $O(n^3)$ arithmetic operations are required for one phase. However, the two major steps of the algorithm, that is the orthogonalization of the basis (respectively, the repair of the orthogonalization) and the size reduction of the whole matrix, are well suited for parallelization. Thus, the following theorem holds.

THEOREM 3.2 (see [3, 4, 9, 13]). *An all-swap phase can be done in $O(n)$ parallel steps* (*according to the number of arithmetic operations and the number of communications of numbers*) *on a mesh-connected network of n^2 processors.*

4. Parallel complexity. In order to achieve a speedup of n^2 using n^2 processors in comparison with the LLL algorithm, it must be shown that a short vector can be found after at most $n \log B$ all-swap phases. The following analysis is taken from [3].

THEOREM 4.1 (parallel time complexity). *After at most $n \log B$ all-swap phases*[1] *the following situation holds for all i with $1 \le i \le n$:*

$$||b_1^*||^2 \cdot ||b_2^*||^2 \cdots ||b_i^*||^2 \le \tfrac{4}{3} \cdot c^{\frac{i(n-i)}{2}} \cdot (\det L)^{\frac{2i}{n}}$$

with $c = \frac{32}{9}$.

Proof. Consider the ratios

$$(1)\quad \frac{||b_1^*||^2}{c^{\frac{n-1}{2}}(\det L)^{\frac{2}{n}}} \cdot \frac{||b_2^*||^2}{c^{\frac{n-3}{2}}(\det L)^{\frac{2}{n}}} \cdots \frac{||b_i^*||^2}{c^{\frac{n-2i+1}{2}}(\det L)^{\frac{2}{n}}} = \frac{\prod_{j=1}^{i} ||b_j^*||^2}{c^{\frac{i(n-i)}{2}}(\det L)^{\frac{2i}{n}}} =: v(i).$$

[1] In the following log denotes the logarithm with basis $\frac{4}{3}$.

We are finished when all these ratios $v(i), 1 \le i \le n$, are less than $\frac{4}{3}$. Let max be the maximum of all these ratios, that is $max = \max\{v(i) | 1 \le i \le n\}$, and let max' be the maximum after one further all-swap phase.

It will be shown that every all-swap phase decreases the value of max by a factor less than $\frac{3}{4}$, that is $max' \le \frac{3}{4} max$, if $max > \frac{4}{3}$. In the beginning, $max \le B^n$ since $||b_i^*||^2 \le B$ holds for all $1 \le i \le n$. Therefore, all ratios are less than $\frac{4}{3}$ after at most $n \log B$ all-swap phases.

PROPOSITION 4.2. *If $max > \frac{4}{3}$ and $v(i) > \frac{3}{4} max$, the product $||b_1^*||^2 \cdot ||b_2^*||^2 \cdots ||b_i^*||^2$ and thereby $v(i)$ is decreased by a factor less than $\frac{3}{4}$ in one all-swap phase. All other ratios remain unchanged or decrease.*

Proof of the proposition. If two vectors b_j and b_{j+1} are swapped, the product $||b_1^*||^2 \cdot ||b_2^*||^2 \cdots ||b_j^*||^2$ is decreased by a factor less than $\frac{3}{4}$; all other products remain unchanged. See [7] for these facts. Thus, we have to show that $max \le \frac{4}{3}$ or b_i and b_{i+1} are swapped for all i with $v(i) > \frac{3}{4} max$ in one all-swap phase; that is, if $v(i) > \frac{3}{4} max$, then $||b_i^*||^2 > 2||b_{i+1}^*||^2$. If this swap condition holds for an even index i, it is also valid after one odd swapping phase.

If $v(n) > \frac{3}{4} max$, then $max < \frac{4}{3}$ since $v(n) = 1$.

Therefore, consider an index $i, 1 \le i \le n-1$, with $v(i) > \frac{3}{4} max$. We show in the following that the vectors b_i and b_{i+1} are swapped in this case.

If $v(i) > \frac{3}{4} max$, then $v(i) > \frac{3}{4} v(i+1)$. Using (1) we obtain

$$\frac{\frac{4}{3} (\det L)^{\frac{2}{n}} c^{\frac{n-2i-1}{2}}}{||b_i^*||^2} > \frac{||b_{i+1}^*||^2}{||b_i^*||^2}. \tag{2}$$

In the same way, if $v(i) > \frac{3}{4} max$ and $i > 1$, then $v(i) > \frac{3}{4} v(i-1)$ holds and therefore

$$||b_i^*||^2 > \frac{3}{4} c^{\frac{n-2i+1}{2}} (\det L)^{\frac{2}{n}}. \tag{3}$$

This relation is also valid for $i = 1$ if $max \ge 1$ and $v(1) > \frac{3}{4} max$.

However, it follows from (2), (3), and the choice of c that

$$\frac{\left(\frac{4}{3}\right)^2}{c} = \frac{1}{2} > \frac{||b_{i+1}^*||^2}{||b_i^*||^2}, \tag{4}$$

and therefore the swap condition is satisfied.

Thus, the proposition and the theorem have been proven. □

In particular, we have found a short vector as shown in the following corollary.

COROLLARY 4.3. *After at most $n \log B$ all-swap phases, the vector b_1 of the resulting lattice basis satisfies*

$$||b_1|| \le \frac{4}{3} c^{\frac{n-1}{4}} (\det L)^{\frac{1}{n}}.$$

Proof. Use Theorem 4.1 with $i = 1$ and $b_1 = b_1^*$. □

COROLLARY 4.4. *A short vector of a lattice can be found in $O(n^2 \log B)$ parallel steps using a network of n^2 processors. Thus, an asymptotical[2] optimal speedup of n^2 can be achieved in comparison with the serial LLL algorithm.*

Proof. See Corollary 4.3 and Theorems 3.2 and 2.4. □

[2] I.e., this holds for $n \to \infty$.

5. Size of the numbers and binary complexity. The time for one operation depends on the size of the numbers. If we start with an integral basis, that is $b_i \in \mathbf{Z}^n$ for all $1 \le i \le n$, all numbers occurring throughout the algorithm are rational with denominators less than B^n [7]. See [7, 5] for an analysis of the sizes of the numbers of the LLL algorithm. The bounds are summarized in Table 1. The known analysis of the parallel algorithm is reviewed in the next section. Then a new result for the parallel case is given.

5.1. Known analysis of the parallel case. In the parallel case the Gram–Schmidt orthogonalization is computed for the whole matrix whereby bigger numbers than in the serial case can occur where the orthogonalization is only computed for the reduced part of the basis.

THEOREM 5.1 (see [7]). *After the computation (or repair) of the Gram–Schmidt orthogonalization we have* $\mu_{i,j}^2 \le iB^n$.

In the parallel algorithm the size reduction is computed simultaneously for all columns. Therefore, the size reduction of each column is performed using nonreduced columns of the basis. This leads to the following theorem.

THEOREM 5.2. *Let* $\mu_{i,j}^2 \le M$ *for all* $1 \le j < i \le n$ *before the size reduction. Then the following inequalities are valid during the size reduction phase of the parallel algorithm (see also* [4, 13]*):*

$$\mu_{i,j}^2 \le 4^n M^n \qquad \text{for} \quad 1 \le j < i \le n, \tag{5}$$

$$\|b_i\|^2 \le nB4^n M^n \qquad \text{for} \quad 1 \le i \le n. \tag{6}$$

Proof. Consider the size reduction of column k. During the reduction of the element $\mu_{k,k-1}$ the computation $\mu_{k,i} \leftarrow \mu_{k,i} - \lceil \mu_{k,k-1} \rfloor \mu_{k-1,i}$ for $i = 1, \ldots, k-1$ is performed. In the case of a parallel execution, column $k-1$ is not size reduced. Therefore, only the estimation $|\mu_{k-1,i}| \le \sqrt{M}$ is possible. Thus, the new value of $|\mu_{k,i}|$ satisfies $|\mu_{k,i}| \le 2M$. The proposition follows by induction and with $\|b_i\|^2 = \sum_{j=1}^{i} \mu_{i,j}^2 \|b_j^*\|^2$. □

Using Theorems 5.1 and 5.2 the following result about the size of the numbers of the parallel lattice basis reduction algorithm can be derived.

COROLLARY 5.3. *During the parallel lattice basis reduction the following inequalities hold for all values of* $\|b_i\|^2$ *and* $\mu_{i,j}^2$ *(see also* [13]*):*

$$\mu_{i,j}^2 \le (4nB^n)^n \qquad \text{for} \quad 1 \le j < i \le n, \tag{7}$$

$$\|b_i\|^2 \le nB(4nB^n)^n \qquad \text{for} \quad 1 \le i \le n. \tag{8}$$

5.2. New results. Two examples for Theorems 5.1 and 5.2 are presented where large numbers do in fact occur. However, both situations never occur simultaneously. Therefore, a better analysis of the size of the numbers can be derived than the one given in Corollary 5.3. This new result is given in Theorem 5.7.

THEOREM 5.4. *The bound given in Theorem* 5.1 *is asymptotically tight.*

Proof. Consider the following $n \times n$ basis for some $D > 1 \in \mathbf{Z}$:

$$\begin{pmatrix} D & D+1 & \cdots & \cdots & D+1 & 0 \\ 0 & D & D+1 & \cdots & D+1 & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & D & D+1 & 0 \\ 1 & \cdots & 1 & 1 & 1 & 1 \\ 0 & \cdots & \cdots & 0 & 0 & 1 \end{pmatrix}.$$

For the Gram–Schmidt orthogonalization of that basis we have

$$\mu_{n,n-1} = (-1)^{n-2} D^{n-2}.$$

The proof can be carried out by induction over n using "Givens" rotations for the computation of the Gram–Schmidt orthogonalization (see, for example, [1] for Givens rotations and [3] for the computation of the Gram–Schmidt orthogonalization via Givens rotations). □

THEOREM 5.5. *The bound for $\mu_{i,j}^2$ given in Theorem* 5.2 *is asymptotically tight.*

Proof. Consider the following $n \times n$ basis for some $D > 1 \in \mathbf{Z}$:

$$(b_1, \ldots, b_n) = \begin{pmatrix} 1 & D & \cdots & D \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & D \\ 0 & \cdots & 0 & 1 \end{pmatrix}.$$

For the matrix of the Gram–Schmidt coefficients of the basis above we have that $M^T = (b_1, \ldots, b_n)$. By induction it can easily be shown that coefficients of the size D^{n-1} occur during the size reduction of column n. □

In order to prove that both cases given above cannot occur simultaneously, we use the following lemma.

LEMMA 5.6. *Consider the size reduction of column k, $2 \le k \le n$, that is of the elements $\mu_{k,k-1}, \ldots, \mu_{k,1}$. After the size reduction of the element $\mu_{k,k-i}$ with $1 \le i \le k-2$, $2^i - 1$ products of Gram–Schmidt coefficients were subtracted from each coefficient $\mu_{k,k-j}$, $j > i$. Each product only consists of (rounded) coefficients of* different *rows of the matrix M^{T} and only of coefficients of the rows $k-1, k-2, \ldots, k-i$ and $k-j$.*

Proof. The proof is done by induction over i.

Let $\mu_{x,y}^{(0)}$ denote $\mu_{x,y}$ before the size reduction of column k, and let $\mu_{x,y}^{(i)}$ denote $\mu_{x,y}$ after the size reduction of $\mu_{k,k-i}$.

$i = 1$: After the size reduction of $\mu_{k,k-1}$ the following is valid for $j > 1$:

$$\mu_{k,k-j}^{(1)} = \mu_{k,k-j}^{(0)} - \lceil \mu_{k,k-1}^{(0)} \rfloor \mu_{k-1,k-j}^{(0)}.$$

Thus, the proposition is true for $i = 1$.

After the size reduction of $\mu_{k,k-i-1}$ the following holds for $j > 1$:

$$\mu_{k,k-j}^{(i+1)} = \mu_{k,k-j}^{(i)} - \lceil \mu_{k,k-i-1}^{(i)} \rfloor \mu_{k-i-1,k-j}^{(0)}.$$

Now, the proposition for $i+1$ follows with the induction hypothesis for $\mu_{k,k-j}^{(i)}$ and $\mu_{k,k-i-1}^{(i)}$. □

Now the following theorem about the size of the numbers of the parallel lattice basis reduction can be shown.

THEOREM 5.7. *During the parallel size reduction phase the following inequalities hold:*

$$\mu_{i,j}^2 \le n^n (2B)^{2n} \qquad \text{for} \quad 1 \le j < i \le n, \tag{9}$$

$$\|b_i\|^2 \le nBn^n (2B)^{2n} \qquad \text{for} \quad 1 \le i \le n. \tag{10}$$

Proof. During the size reduction only sums with at most 2^n terms occur. Each term is a product of coefficients of *different* rows. With $\mu_{i,j}^2 \le \frac{\|b_i\|^2}{\|b_j^*\|^2}$ [7] we have

$$\prod \mu_{i,j}^2 \le \prod \frac{\|b_i\|^2}{\|b_j^*\|^2}.$$

With $\|b_i\|^2 \le nB$ [7] it follows that

$$\prod \|b_i\|^2 \le (nB)^n.$$

As $\|b_1^*\|^2 \cdots \|b_n^*\|^2 \ge 1$ and $\|b_i^*\|^2 \le B$ for all $1 \le i \le n$ (see [7]) and because in each product only different b_j^*'s occur we obtain

$$\prod \|b_j^*\|^2 \ge \frac{1}{B^n},$$

and therefore for each product the following inequalities are valid

$$\prod \mu_{i,j}^2 \le n^n B^{2n},$$
$$\left|\prod \mu_{i,j}\right| \le n^{n/2} B^n.$$

Each sum consists of at most 2^n products. Therefore we obtain for all values

$$|\mu| \le n^{n/2} (2B)^n,$$
$$\mu^2 \le n^n (2B)^{2n}.$$

Again, the proposition for the values of b_i follows with $\|b_i\|^2 = \sum_{j=1}^{i} \mu_{i,j}^2 \|b_j^*\|^2$. ∎

Table 1 summarizes the results with respect to the sizes of the numbers.

TABLE 1
Comparison of the sizes of the numbers.

	LLL [7, 5]	Parallel (old analysis)	Parallel (new analysis)
$\mu_{i,j}^2$ after Gram–Schmidt	$n2^n B$	nB^n	nB^n
$\|b_i\|^2$ after Gram–Schmidt	nB	nB	nB
$\mu_{i,j}^2$ during size reduction	$n2^{3n} B$	$(4nB^n)^n$	$n^n (2B)^{2n}$
$\|b_i\|^2$ during size reduction	$n^2 2^{3n} B^2$	$nB(4nB^n)^n$	$nBn^n (2B)^{2n}$

5.3. Further remarks. There is a special class of lattices (which are constructed for the solution of the subset sum problem) for which a better analysis of the size of the numbers in the parallel case is possible [13].

Joux [4] has presented a method for avoiding big numbers during the parallel size reduction. But also with that method a better analysis than the one given in Theorem 5.7 is not possible.

In practical experiments for the parallel algorithm of [13] and [2, 3] larger numbers than in the serial algorithm do not occur.

5.4. Binary complexity. The estimation of the sizes of the numbers and a detailed analysis by Kaltofen [5] lead to the following binary complexity of the LLL algorithm using standard arithmetic.

THEOREM 5.8 (see [5]). *The LLL algorithm has binary complexity* $O(n^5(\log B)^3 + n^6(\log B)^2)$.

The binary parallel complexity is given by the following theorem.

THEOREM 5.9. *The binary complexity of parallel lattice basis reduction using* n^2 *processors is* $O(n^4(\log B)^3)$.

Proof. The size of all numbers (numerators and denominators) is bounded by $O(n \log B)$. Thus the time for one arithmetic operation is at most $O(n^2(\log B)^2)$ using standard arithmetic. At most $O(n^2 \log B)$ arithmetic operations must be performed. □

Acknowledgments. The authors would like to thank the two unnamed referees for their helpful remarks and suggestions.

REFERENCES

[1] G. GOLUB AND C. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[2] C. HECKLER, *Automatische Parallelisierung und parallele Gitterbasisreduktion*, dissertation, Institut für Mikroelektronik, Universität des Saarlandes, Saarbrücken, Germany, March 1995.

[3] C. HECKLER AND L. THIELE, *A parallel lattice basis reduction for mesh-connected processor arrays and parallel complexity*, in Proc. SPDP'93, Dallas, TX, December 1–4, 1993, pp. 400–407.

[4] A. JOUX, *A fast parallel lattice reduction algorithm*, in Proc. Second Gauss Symposium, Munich, 1993.

[5] E. KALTOFEN, *On the complexity of finding short vectors in integer lattices*, in Proc. EUROCAL83, London, LNCS 162, Springer-Verlag, Berlin, New York, 1983, pp. 236–244.

[6] R. KANNAN, *Algorithmic geometry of numbers*, Ann. Rev. Comput. Sci., 16 (1987), pp. 231–267.

[7] A. K. LENSTRA, H. W. LENSTRA JR., AND L. LOVASZ, *Factoring polynomials with rational coefficients*, Math. Ann., 261 (1982), pp. 515–534.

[8] H. W. LENSTRA JR., *Integer programming with a fixed number of variables*, Math. Oper. Res., 8 (1983), pp. 538–548.

[9] J. ROCH AND G. VILLARD, *Parallel gcd and lattice basis reduction*, in Proc. CONPAR92, Lyon, LNCS 632, Springer-Verlag, Berlin, New York, 1992, pp. 557–564.

[10] C. SCHNORR, *A more efficient algorithm for lattice basis reduction*, J. Algorithms, 9 (1988), pp. 47–62.

[11] C. SCHNORR AND M. EUCHNER, *Lattice basis reduction: Improved practical algorithms and solving subset sum problems*, in Proc. FCT'91, Gosen, Germany, LNCS 529, Springer-Verlag, Berlin, New York, 1991, pp. 68–85.

[12] A. SCHRIJVER, *Theory of Linear and Integer Programming*, John Wiley and Sons, New York, 1986.

[13] G. VILLARD, *Parallel lattice basis reduction*, in Proc. International Symposium on Symbolic and Algebraic Computation, Berkeley, CA, ACM Press, New York, 1992, pp. 269–277.

SIAM J. COMPUT.
Vol. 27, No. 5, pp. 1303–1333, October 1998
© 1998 Society for Industrial and Applied Mathematics
007

ON THE FAULT TOLERANCE OF SOME POPULAR BOUNDED-DEGREE NETWORKS*

F. THOMSON LEIGHTON†, BRUCE M. MAGGS‡, AND RAMESH K. SITARAMAN§

Abstract. In this paper, we analyze the fault tolerance of several bounded-degree networks that are commonly used for parallel computation. Among other things, we show that an N-node butterfly network containing $N^{1-\epsilon}$ worst-case faults (for any constant $\epsilon > 0$) can emulate a fault-free butterfly of the same size with only constant slowdown. The same result is proved for the shuffle-exchange network. Hence, these networks become the first connected bounded-degree networks known to be able to sustain more than a constant number of worst-case faults without suffering more than a constant-factor slowdown in performance. We also show that an N-node butterfly whose nodes fail with some constant probability p can emulate a fault-free network of the same type and size with a slowdown of $2^{O(\log^* N)}$. These emulation schemes combine the technique of redundant computation with new algorithms for routing packets around faults in hypercubic networks. We also present techniques for tolerating faults that do not rely on redundant computation. These techniques tolerate fewer faults but are more widely applicable because they can be used with other networks such as binary trees and meshes of trees.

Key words. fault tolerance, network emulation, butterfly network

AMS subject classifications. 68M07, 68M10, 68M15, 68Q68

PII. S0097539793255163

1. Introduction.

In this paper, we analyze the effect of faults on the computational power of bounded-degree networks such as the butterfly network, the shuffle-exchange network, and the mesh of trees. The main objective of our work is to devise methods for circumventing faults in these networks using as little overhead as possible and to prove lower bounds on the effectiveness of optimal methods. We consider both worst-case and random fault patterns, and we always assume that faulty components are totally disabled (e.g., a faulty node cannot be used to transport a packet of data through the network). We also assume that the faults in a network are static and detectable and that information concerning the location of faults can be used when reconfiguring the network to circumvent the faults. For simplicity, we restrict our attention to node faults since an edge fault can always be simulated by disabling the node at each end of the edge.

There are several ways to measure the effect of faults on a network. In this paper, we are primarily concerned with the amount by which a collection of faults can slow down some computation on the network. For example, if a butterfly network

* Received by the editors September 9, 1993; accepted for publication (in revised form) July 20, 1996; published electronically May 19, 1998.
http://www.siam.org/journals/sicomp/27-5/25516.html

† Mathematics Department and Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 (ftl@math.mit.edu). This author was supported in part by Army contract DAAH04-95-0607 and by ARPA contract N00014-95-1-1246.

‡ School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213 (bmm@cs.cmu.edu). This author was supported in part by the Air Force Material Command (AFMC) and ARPA under contract F196828-93-C-0193, by ARPA contracts F33615-93-1-1330 and N00014-95-1-1246, and by an NSF National Young Investigator Award, CCR-94-57766, with matching funds provided by NEC Research Institute. This research of this author was conducted while the author was employed at NEC Research Institute.

§ Department of Computer Science, University of Massachusetts, Amherst, MA 01003 (ramesh@cs.umass.edu). This author was supported in part by NSF grant CCR–94–10077. The research of this author was conducted while the author was a student at Princeton University.

containing some faults is to be used for packet routing, we are concerned with how many nodes can send and receive packets, and how much longer it takes the faulty butterfly to deliver all of the packets than it takes a fault-free butterfly to perform the same task. More generally, we are interested in the length of time it takes an impaired network to emulate a single step of a fault-free network of the same type and size. In particular, we define the *slowdown* caused by a set of faults in a network G to be the minimum value of S such that any computation that takes T steps on G when there are no faults can be performed in at most $S \cdot T$ steps on G when faults are present. One of our main goals is to understand the relationship between slowdown and the number of faults for commonly used networks. In particular, we prove bounds on the number of faults that can be tolerated without losing more than a constant factor in speed.

We have two approaches for emulating a fault-free network G on an isomorphic faulty network H. The first approach is to find an embedding of G into H that avoids the faults in H. The second approach uses redundant computation; i.e., we allow H to emulate some of the nodes of G in more than one place. At first glance this approach seems disadvantageous, since H ends up performing more work. As we shall see, however, the freedom to emulate a node of G in more than once place allows H to have results ready where and when they are needed and can greatly reduce the slowdown of the emulation.

1.1. Emulations based on embeddings. An *embedding* maps the nodes of G to nonfaulty nodes of H and the edges of G to nonfaulty paths in H. A good embedding is one with minimum load, congestion, and dilation, where the *load* of an embedding is the maximum number of nodes of G that are mapped to any single node of H, the *congestion* of an embedding is the maximum number of paths that pass through any edge e of H, and the *dilation* of an embedding is the length of the longest path. The load, congestion, and dilation of the embedding determine the time required to emulate each step of G on H. In particular, Leighton, Maggs, and Rao have shown [30] that if there is an embedding of G in H with load l, congestion c, and dilation d, then H can emulate any computation on G with slowdown $O(l + c + d)$.

In this paper, we are most interested in embeddings for which the load, congestion, and dilation are all constant (independent of the size of the network). In particular, we show in section 2 how to embed a fault-free N-input butterfly into an N-input butterfly containing $\log^{O(1)} N$ worst-case faults using constant load, congestion, and dilation. A similar result is also proved for the N-node mesh of trees. Hence, these networks can tolerate $\log^{O(1)} N$ worst-case faults with constant slowdown.

Previously, no connected bounded-degree networks were known to be able to tolerate more than a constant number of worst-case faults without suffering more than a constant-factor loss in performance. Indeed, it was only known that

1. any embedding of an N-node (two- or three-dimensional) array into an array of the same size containing more than a constant number of worst-case faults must have more than constant load, congestion, or dilation [22, 25, 31], and
2. the N-node hypercube can be reconfigured around $\log^{O(1)} N$ worst-case faults with constant load, congestion, and dilation [2, 12].

The embeddings that we use in section 2 are level preserving; i.e., nodes in a particular level of the fault-free network are mapped to nodes on the same level of the faulty network. We take a significant step toward proving the limitation of embedding techniques for the emulation of these networks by showing that no level-preserving embedding strategy with constant load, congestion, and dilation can tolerate more

than $\log^{O(1)} N$ worst-case faults. Whether or not there is a natural low-degree N-node network (the hypercube included) that can be reconfigured around more than $\log^{O(1)} N$ faults with constant load, congestion, and dilation using (not necessarily level-preserving) embedding techniques remains an interesting open question.

1.2. Fault-tolerant routing algorithms. In section 3, we shift our attention to the routing capabilities of hypercubic networks containing faults. The algorithms developed in this section are later used by the emulation schemes based on redundant computation. First we prove in section 3.1 that an N-input butterfly with f worst-case faults can support an $O(\log N)$-step randomized packet routing algorithm for the nodes in $N - 6f$ rows of the butterfly. The ability of the butterfly to withstand faults in this context is important because butterflies are often used solely for their routing abilities. Previously, it was known that expander-based multibutterfly networks can tolerate large numbers of worst-case faults without losing their routing powers [6, 28], but no such results were known for butterflies or other hypercubic networks. A corollary of this result is that an N-input butterfly with $N/12$ worst-case faults can support an $O(\log N)$-step randomized routing algorithm for a majority of its nodes. Note that the number of faults is optimal to within a constant factor, since it is possible to partition an N-input butterfly into connected components of size $O(\sqrt{N} \log N)$ with N faults. In section 3.2 we show that butterflies with faults can also be used for circuit switching. In particular, we show that even if a $2N$-input $O(1)$-dilated Beneš network contains $N^{1-\epsilon}$ worst-case faults (for any $\epsilon > 0$), there is still a set of $2N - o(N)$ inputs I and a set of $2N - o(N)$ outputs O such that for any one-to-one mapping $\phi : I \mapsto O$ it is possible to route edge-disjoint paths from i to $\phi(i)$ for all $i \in I$. This result substantially improves upon previous algorithms for fault-tolerant circuit switching in Beneš networks [41, 49], which dealt with a constant number of faults by adding an extra stage to the network.

1.3. Emulations using redundant computation. In section 4, we use the fault-tolerant routing algorithm from section 3.2 to show that an N-input butterfly with $N^{1-\epsilon}$ worst-case faults (for any constant $\epsilon > 0$) can emulate a fault-free butterfly of the same size with only constant slowdown. A similar result is proved for the shuffle-exchange network. These results are stronger than the reconfiguration results proved in section 2 because the number of faults tolerated is much larger. The approach used in section 4 differs from the embedding-based approaches in section 2 in that a single node of the fault-free butterfly is emulated by (possibly) several nodes in the faulty butterfly. Allowing redundant computation provides greater flexibility when embedding one network in another (thereby attaining greater fault tolerance) but also adds the complication of ensuring that replicated computations stay consistent (and accurate) over time. This technique was previously used in the context of (fault-free) work-preserving emulations of one network by another [19, 26, 38, 39, 40, 47].

The techniques developed in section 4 also have applications for hypercubes. For example, in section 4.4, we use them to show than an N-node hypercube with $N^{1-\epsilon}$ worst-case faults can emulate T steps of any normal algorithm [27] in $O(T + \log N)$ time. (The set of normal algorithms include FFT, bitonic sort, and other important ascend–descend algorithms.) Previously, such results were known only for hypercubes containing $\log^{O(1)} N$ faults [2, 12, 13]. Whether or not an N-node hypercube can tolerate more than $\log^{O(1)} N$ faults with constant slowdown for general computations remains an important unresolved question.

In section 5, we show that even if each node in an N-input butterfly fails indepen-

dently with probability $p = 1/\log^{(k)} N$, where $\log^{(k)}$ denotes the logarithm function iterated k times, the faulty butterfly can still emulate a fault-free N-input butterfly with slowdown $2^{O(k)}$, with high probability. For $k = O(\log^* N)$ the node failure probability is constant, and the slowdown is $2^{O(\log^* N)}$, which grows very slowly with N. Whether or not this result can be improved remains an interesting open question. Until recently, no results along these lines were known for the butterfly, unless routing is allowed through faulty nodes [5], which simplifies matters substantially. Tamaki [51] has recently discovered an emulation scheme with slowdown $O((\log\log N)^{8.2})$. He also introduced a class of bounded-degree networks called cube-connected arrays [52] and showed that an N-node network in this class with constant-probability random faults can emulate itself with expected slowdown approximately $\log\log N$. These networks can also tolerate up to $\log^{O(1)} N$ worst-case faults with approximately $\log\log N$ slowdown.

1.4. Additional previous work. There is a substantial body of literature concerning the fault tolerance of communication networks. We do not have the space to review all of this literature here, but we would like to cite the papers that are most relevant. In particular, [2, 5, 9, 14, 23, 24, 25, 35, 44, 52] show how to reconfigure a network with faults so that it can emulate a fault-free network of the same type and size. A fault-tolerant area-universal network is presented in [53]. References [4, 10, 11, 16, 17] show how to design a network H that contains G as a subnetwork even if H contains some faults. Algorithms for routing messages around faults appear in [1, 6, 8, 15, 24, 25, 28, 34, 36, 41, 43, 44, 49]. The fault-tolerance of sorting networks is studied in [7, 32]. Finally, [12, 56, 57] show how to perform certain computations in hypercubes containing faults.

1.5. Network definitions. In this section, we review the structure of some of the networks that we study in this paper. In all of these networks, the edges are assumed to be undirected (or bidirectional).

An N-input $(\log N)$-dimensional *butterfly* network has $N(\log N + 1)$ nodes arranged in $(\log N) + 1$ levels.[1] An 8-input butterfly is shown in Figure 1.1. Each node in the butterfly has a distinct label (w, i), where i is the *level* of the node $(0 \le i \le \log N)$ and w is a $(\log N)$-bit binary number that denotes the *row* of the node. All edges connect pairs of nodes on adjacent levels. Each node (w, i) is connected by a *straight edge* to node $(w, i + 1)$, provided that $i < \log N$. In the figure, straight edges are drawn horizontally. Each node (w, i) is also connected by a *cross edge* to node $(w', i + 1)$, where w and w' differ only in the bit in position i, provided that $i < \log N$. (The most significant bit is in position 0, and the least significant is in position $(\log N) - 1$.) In the figure, cross edges are drawn diagonally. The nodes in level 0 are called the *inputs* of the butterfly, and those in level $\log N$ are called the *outputs*. Sometimes the input and output nodes in each row are assumed to be the same node. In this case, the butterfly has only $N \log N$ nodes. Our results hold whether or not the butterfly wraps around in this way.

In an N-node *hypercube*, each node is labeled with a distinct $(\log N)$-bit binary number. Two nodes in the hypercube are connected by an edge if and only if their labels differ in exactly one bit. The hypercube is the only network considered in this paper in which the degree of each node is not constant.

As in the N-node hypercube, each node in an N-node *shuffle-exchange* network is labeled with a distinct $(\log N)$-bit binary number. An 8-node shuffle-exchange

[1] Throughout this paper, log denotes the base-2 logarithm function, $\log_2$.

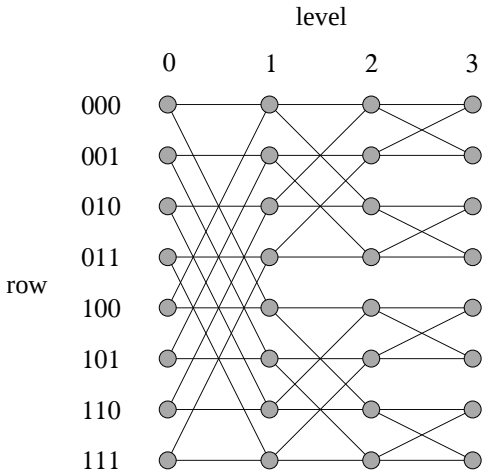


FIG. 1.1. *An* 8*-input butterfly network.*

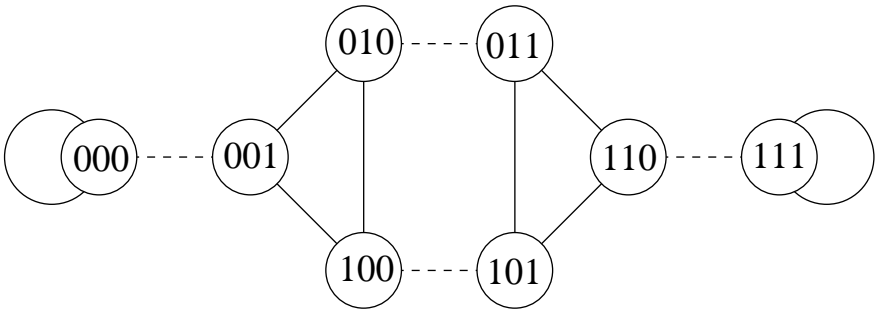


FIG. 1.2. *An* 8*-node shuffle-exchange network.*

network is shown in Figure 1.2. In the shuffle-exchange network, a node labeled u is connected by an *exchange* edge to the node labeled u', where u and u' differ only in the bit in the least significant position (position $(\log N)-1$). Node u is also connected by *shuffle* edges to the nodes labeled u_l and u_r, where u_l and u_r are the one-bit left and right cyclic shifts of u. (If $u_l = u_r$ then there is only one shuffle edge.) In the figure exchange edges are dotted and shuffle edges are solid.

An $N \times N$ *mesh of trees* network [27] is formed by first arranging N^2 nodes (but no edges) in a grid of N rows and N columns. Then for each row, an N-leaf complete binary tree, called a *row tree*, is added. The leaves of the row tree are the nodes of the corresponding row. Similarly, for each column an N-leaf *column tree* is added. The leaves of the column tree are the nodes of the column. Hence, the node at position (i, j) in the grid is a leaf of the ith row tree and jth column tree for $0 \le i, j \le N-1$.

A *circuit-switching network* is used to establish edge-disjoint paths (called circuits) between its inputs and outputs. We call the nodes in a circuit-switching network *switches* to signify that they are used only for routing and not for performing computation. Each switch in a circuit-switching network has a set of incoming edges and a set of outgoing edges. Inside the switch, the incoming edges can be connected to the outgoing edges in any one-to-one fashion. The switches at the first level of the network are called the *input switches*. The switches at the last level are called the

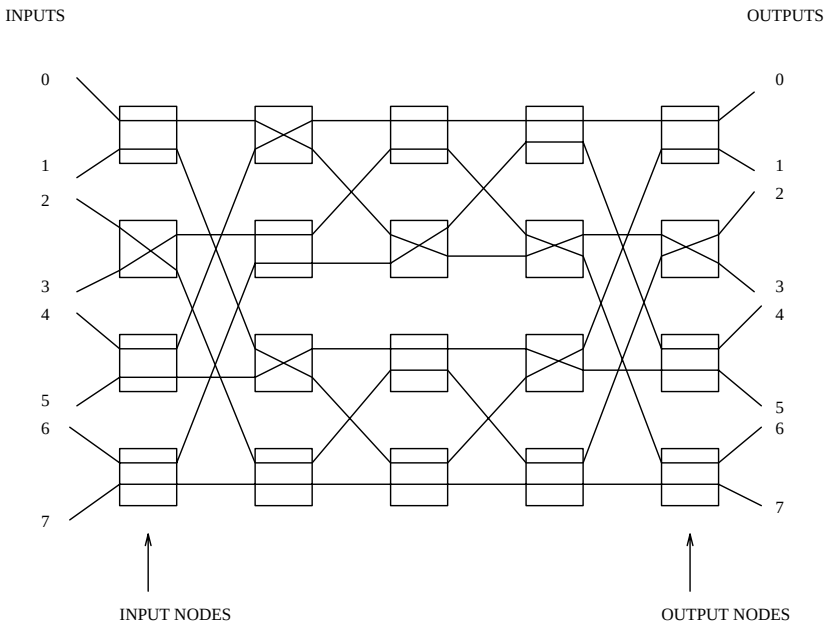


FIG. 1.3. *An 8-input (two-dimensional) Beneš network.*

output switches. The edges into each input switch are called *input edges*, or *inputs*. The edges out of each output switch are called *output edges*, or *outputs*. (Note, however, that in a butterfly network, we use the terms "inputs" and "outputs" to refer to nodes, not edges.) By setting the connections inside the switches, each input edge can be connected to an output edge via a path through the network.

A circuit-switching network with N inputs and N outputs is said to be *rearrangeable* if for any one-to-one mapping ϕ from the inputs to the outputs it is possible to construct edge-disjoint paths in the network connecting the ith input to the $\phi(i)$th output for $0 \leq i \leq N-1$.

The *Beneš* network is a classic example of a rearrangeable network. A $(\log N)$-dimensional Beneš network has $2N$ inputs and $2N$ outputs. Its switches are arranged in $2\log N + 1$ levels of N switches each. The first and last $\log N + 1$ levels each form a $(\log N)$-dimensional butterfly. Hence a Beneš network consists of two back-to-back butterflies sharing level $\log N$. We refer to the switches in levels 0, $\log N$, and, $2\log N$ as the *input switches*, *middle switches*, and *output switches*, respectively. Figure 1.3 shows an 8-input Beneš network in which the inputs are connected to the outputs according to the following mapping ϕ: $\phi(0) = 3$, $\phi(1) = 1$, $\phi(2) = 2$, $\phi(3) = 6$, $\phi(4) = 0$, $\phi(5) = 5$, $\phi(6) = 4$, $\phi(7) = 7$.

2. Emulation by embedding. In this section, we show how to embed a fault-free binary tree, butterfly, or mesh of trees into a faulty network of the same type and size with constant load, congestion, and dilation. As noted in the introduction, finding a constant load, congestion, and dilation embedding is the simplest way of emulating arbitrary computations of a fault-free network on a faulty network of the same type and size with only constant slowdown. We first consider embedding a complete binary tree in a complete binary tree with faults only at its leaves. This result also holds for fat-trees [21, 33] with faults at the leaves. We use this result to

find reconfigurations of butterflies and meshes of trees in which faults may occur at any node. The main result of this section is a proof that an N-node butterfly or mesh of trees network can tolerate $\log^{O(1)} N$ worst-case faults and still emulate a fault-free network of the same type and size with only constant slowdown.

2.1. The binary tree. In Theorem 2.1.2, we show that a fault-free 2^n-leaf complete binary tree can be embedded in another 2^n-leaf complete binary tree containing $S(n,b)$ or fewer faults at its leaves with load and congestion at most 2^b and dilation 1, where $S(n,b)$ is defined for $n \geq b \geq 0$ by the recurrence

$$S(n,b) = S(n-1,b) + S(n-1,b-1) + 1$$

for $n > b > 0$, with boundary conditions $S(n,0) = 0$ and $S(n,n) = 2^n - 1$ for $n \geq 0$. The following lemma provides a useful bound on the growth of $S(n,b)$.

LEMMA 2.1.1. *For all* $n \geq b \geq 1$, $\binom{n}{b} \leq S(n,b) \leq \binom{n+b}{b}$.

Proof. The proof is by induction on n. For $n = 1$, the only possible value of b is 1. In this case, $S(1,1) = 1$ and $\binom{1}{1} = 1 < \binom{2}{1}$. For $n > 1$, there are three cases to consider. First, for $b = 1$, $S(n,1) = n$, and $\binom{n}{1} = n < \binom{n+1}{1}$. Second, for $n = b$, $S(n,n) = 2^n - 1$ and $\binom{n}{n} \leq 2^n - 1 < \binom{2n}{n}$. Finally, for $n > b > 1$, the inequalities are proved inductively using the fact that $\binom{x}{y} = \binom{x-1}{y} + \binom{x-1}{y-1}$, for all $x > y > 0$, and $\binom{x}{y} + 1 \leq \binom{x+1}{y}$ for $x \geq y > 0$. ∎

Using the inequalities $(x/y)^y \leq \binom{x}{y} \leq (xe/y)^y$ for $x \geq y > 0$, we see that for any constant $b > 0$, $S(n,b) = \Theta(n^b)$.

THEOREM 2.1.2. *Given a* 2^n*-leaf complete binary tree* T *with a set of at most* $S(n,b)$ *worst-case faults at the leaves, where* $n \geq b \geq 0$*, it is possible to embed a fault-free* 2^n*-leaf complete binary tree* T' *in* T *so that*

1. *nodes on level* i *of* T' *are mapped to nonfaulty nodes on level* i *of* T*, for* $0 \leq i \leq n$;
2. *the congestion and the load of the embedding are at most* 2^b; *and*
3. *the dilation of the embedding is* 1.

Proof. The proof is by induction on n. For $n = 0$, the only possible value of b is 0 and $S(0,0) = 0$. In this case T is a single fault-free node and T' can be embedded in T with load 1, congestion 0, and dilation 0. For $n > 0$, there are three cases to consider. First, for $b = 0$, $S(n,0) = 0$, so T has no faults. In this case, T' can be embedded in T with load 1, congestion 1, and dilation 1. Second, for $b = n$, $S(n,n) = 2^n - 1$, so there is a single nonfaulty leaf l in T. In this case, all of the 2^n leaves of T' are mapped to l, and the rest of the tree is mapped to the path from the root of T to l. The embedding has load 2^n, congestion 2^n, and dilation 1. Finally, suppose that $n > b > 0$. If both 2^{n-1}-leaf subtrees of T have at most $S(n-1,b)$ faulty leaves, then we use the result inductively in both subtrees. Otherwise, if one 2^{n-1}-leaf subtree (say the left subtree) has $S(n-1,b)+1$ or more faults, then by the definition of $S(n,b)$ the other subtree (the right subtree) has at most $S(n-1,b-1)$ faulty leaves. Hence we can use induction to embed a 2^{n-1}-leaf complete binary tree on the right subtree with dilation 1 and load and congestion 2^{b-1}. By doubling the congestion and the load, we can embed two 2^{n-1}-leaf complete binary trees in the right subtree. This means that we can embed T' in T with dilation 1 and load and congestion 2^b using only the root and the right subtree of T. ∎

Rewriting the number of leaves as $N = 2^n$, we see that for any constant $b > 0$, it is possible to embed a fault-free N-leaf complete binary tree T' in an N-leaf

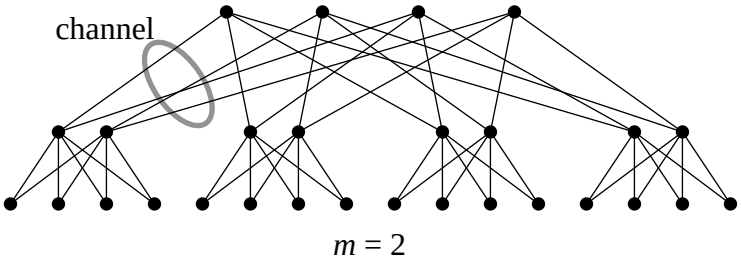


FIG. 2.1. *A 4-ary fat-tree network with depth $m = 2$ in which $r_0 = 4$, $r_1 = 2$, and $r_2 = 1$.*

complete binary tree T containing $S(\log N, b) = \Theta(\log^b N)$ faults with constant load, congestion, and dilation.

This result can be extended to a class of networks called *fat-trees*. A fat-tree of depth m is specified by a sequence of numbers $r_0, r_1, \ldots, r_m$, where $r_m = 1$. (Typically $r_0 \geq r_1 \geq \cdots \geq r_m$.) A fat-tree of depth 0 is a single node, which is both the root node and the leaf node of the tree. A fat-tree of depth m is constructed as follows. At the root of the fat-tree there is a set of r_0 nodes. The subtrees of the root are identical and are constructed recursively. Each is a fat-tree of depth $m - 1$ with number sequence $r_1, \ldots, r_m$. The r_0 root nodes of the fat-tree are connected to the r_1 root nodes of each subtree by a *channel* of edges. There may be any number of edges in the channel, and they may form any pattern of connections, but the channels to each subtree must be isomorphic. Figure 2.1 shows a fat-tree in which $r_0 = 4$, $r_1 = 2$, and $r_2 = 1$. In this figure, the root has four subtrees, as do the roots of these subtrees. Hence the figure shows a 4-ary fat-tree. This fat-tree was chosen for the figure because a fat-tree of this form has been shown to be area universal [33, 21, 30]. Corollary 2.1.3 is stated for binary fat-trees (i.e., fat-trees in which the root has two subtrees), but similar results can be proven for 4-ary fat-trees.

COROLLARY 2.1.3. *A $(\log N)$-depth binary fat-tree can be embedded in a level-preserving fashion in an isomorphic fat-tree with $S(\log N, b)$ worst-case faults at its leaves with load and congestion 2^b and dilation 1.*

Proof. Associate the nodes of the fat-tree with the nodes of an N-node complete binary tree as follows. Associate all root nodes of the fat-tree with the root of the complete binary tree. Recursively associate the nodes of the left (right) subtree of the fat-tree with the nodes of the left (right) subtree of the complete binary tree. Note that every leaf of the fat-tree is associated with a distinct leaf of the complete binary tree. Given a fat-tree F with some faulty leaves, let T be a complete binary tree whose leaf is faulty if and only if the corresponding leaf in F is faulty. A fault-free complete binary tree T' can be embedded in T by embedding subtrees of T' into subtrees of T using the procedure given in Theorem 2.1.2. The same embedding can be used to embed a fault-free fat-tree F' in F by embedding the corresponding subtrees of F' into the corresponding subtrees of F. The dilation and load are the same as that of the complete binary tree embedding. □

COROLLARY 2.1.4. *A* $(\log N)$*-dimensional butterfly can be embedded in a level-preserving fashion in an isomorphic butterfly with* $S(\log N, b)$ *worst-case faults at level* $\log N$ *with load and congestion* 2^b *and dilation* 1.

Proof. A $(\log N)$-dimensional butterfly is a binary fat-tree of depth $\log N$ with $r_i = 2^{\log N - i}$. The leaves of the fat-tree are the nodes in level $\log N$ of the butterfly. □

2.2. The mesh of trees and the butterfly. We can use Theorem 2.1.2 to show that the mesh of trees and the butterfly network can tolerate $\log^{O(1)} N$ worst-case faults with constant slowdown, even when a fault can occur at *any node* of the network. The proof uses the fact that both the mesh of trees and the butterfly can be viewed as a special kind of product graph, which we call an *external product graph*.

As external product graph is defined as follows. Let G be a graph in which some set of N nodes have been labeled as *external* nodes. For example, if G is a tree, the leaves could be the external nodes, or, if G is a butterfly, the nodes on level $\log N$ (the outputs) could be the external nodes. Given a graph G with N external nodes, the external product graph of G (denoted PG) is constructed as follows. Make $2N$ copies of G, $G_{i,j}$ for $i = 1, 2$ and $0 \leq j \leq N-1$. Number the external nodes of each copy from 0 to $N-1$. Now identify the kth external node in $G_{1,j}$ with the jth external node in $G_{2,k}$ for all $0 \leq j, k \leq N-1$. (By "identify" we mean make them the same node of the graph PG.) The resulting graph is the external product graph of G. As an example, when the graph G is a tree and its leaves are the external nodes, the graph PG is a mesh of trees network. As another example, when G is a butterfly and its outputs are external nodes, PG is a butterfly with twice the dimension.

We now show that if G can tolerate faults in its external nodes, then PG can tolerate faults at any of its nodes.

THEOREM 2.2.1. *If a graph* G' *can be embedded in a level-preserving fashion with load* l*, congestion* c*, and dilation* d *in an isomorphic graph* G *with* f *worst-case faults located in its external nodes, then it is possible to embed the product graph* PG' *in a level-preserving fashion with load* l^2*, congestion* lc*, and dilation* d *in an isomorphic graph* PG *with* $f/2$ *worst-case faults located in any of its nodes.*

Proof. Let PG and PG' be made up of graphs isomorphic to G called $G_{i,j}$ and $G'_{i,j}$, respectively, for $i = 1, 2$ and $1 \leq j \leq N$. Let CG and CG' also be graphs isomorphic to G. The jth external node of CG is declared to be faulty if and only if either $G_{1,j}$ or $G_{2,j}$ contains a fault. If PG has $f/2$ faults, then CG has at most f faults (since an external node of PG can appear both as the kth leaf of $G_{1,j}$ and as the jth leaf of $G_{2,k}$). Let Φ be a level-preserving embedding of the fault-free graph CG' into CG with load l, congestion c, and dilation d and define ϕ so that Φ maps the jth external node of CG' to the $\phi(j)$th external node of CG. We embed PG' into PG by mapping $G'_{i,j}$ to $G_{i,\phi(j)}$ using Φ to map the individual nodes of $G'_{i,j}$ to $G_{i,\phi(j)}$ for $i = 1, 2$ and $0 \leq j \leq N-1$. (Hence, the mapping Φ is used twice.) It follows from the definition of faults in CG that $G_{i,\phi(j)}$ is fault-free. Therefore, no nodes of PG' are mapped to faulty nodes of PG. We need to verify that our mapping is well defined, i.e., that it doesn't map an external node of PG' to more than one node of PG. The kth external node of $G'_{1,j}$ is the same as the jth external node of $G'_{2,k}$. The former is mapped to the $\phi(k)$th external node of $G_{1,\phi(j)}$ and the latter to the $\phi(j)$th external node of $G_{2,\phi(k)}$. These nodes are the same node of PG. Hence, the mapping is well defined. The dilation of the mapping is d. The number of copies $G'_{i,j}$ of PG' mapped to any particular $G_{i,\phi(j)}$ is at most l. Each copy can map l nodes onto any particular node of $G_{i,\phi(j)}$. Therefore, the load is at most l^2, and the congestion is at most lc. □

Theorem 2.2.1 is readily applied to the butterfly and mesh of trees networks. For simplicity, we state the result for a two-dimensional mesh of trees. The same techniques, however, can be used to show that any constant-dimension mesh of trees can tolerate $\log^{\Theta(1)} N$ worst-case faults with only constant slowdown.

THEOREM 2.2.2. *A $2 \log N$-dimensional butterfly can be embedded in a level-preserving fashion in a $2 \log N$-dimensional butterfly containing $S(\log N, b) = \Theta(\log^b N)$ worst-case faults with load and congestion 2^{2b} and dilation 1.*

Proof. The proof follows from Corollary 2.1.4 and Theorem 2.2.1. □

THEOREM 2.2.3. *An $N \times N$ mesh of trees can be embedded in a level-preserving fashion in an $N \times N$ mesh of trees containing $S(\log N, b) = \Theta(\log^b N)$ worst-case faults with load and congestion 2^{2b} and dilation 1.*

Proof. The proof follows from Theorems 2.1.2 and 2.2.1. □

The results of this subsection can also be shown by using the fact that the butterfly and the mesh of trees can be expressed as the layered cross product [18] of two complete binary trees (or variations thereof) [3] and proving a theorem analogous to Theorem 2.2.1 for layered cross product graphs.

2.3. Limitations of level-preserving embeddings. We do not know whether Theorems 2.1.2, 2.2.2, and 2.2.3 can be improved if the level-preserving constraint is removed. However, we can show that the bounds in Theorems 2.1.2, 2.2.2, and 2.2.3 are tight if the embedding is forced to be level preserving. The proof uses a construction called an arrow diagram.

Given an N-leaf binary tree T with faults at its leaves, an *arrow diagram* has arrows drawn from some nodes of T to their siblings, with no pairs of antiparallel arrows allowed. We define a *b-legal* arrow diagram as follows.

1. On any path from the root to a faulty leaf, there is an arrow from a node on the path to a node not on the path (called an outgoing arrow).
2. On any path with no outgoing arrow, there can be at most b incoming arrows.

An arrow diagram is called *legal* if it is b-legal for any $0 \leq b \leq n$.

Suppose that an adversary is allowed to place faults at the leaves of a 2^n-leaf complete binary tree. Let $T(n, b) + 1$ be the minimum number of faults needed by the adversary to make it impossible to construct a b-legal arrow diagram for the tree. Note that if a diagram is illegal for some set of faults then it cannot be made legal by adding another fault. Hence allowing more faults only makes the adversary more powerful. We bound the value of $T(n, b)$ as follows.

LEMMA 2.3.1. *For $n \geq b \geq 0$,*

$$T(n,b) \leq \begin{cases} 0 & \text{for } b = 0, \\ T(n-1,b) + T(n-1,b-1) + 1 & \text{for } 0 < b < n, \\ 2^n - 1 & \text{for } b = n. \end{cases}$$

Proof. First, suppose that $b = 0$. If the tree has one or more faults, then any legal arrow diagram must have at least one arrow. If the diagram has at least one arrow, then there must be a path from the root of the tree to a leaf having at least one incoming arrow and no outgoing arrow. Such a path can be recursively constructed as follows. Choose the arrow that is closest to the root of the tree, and let this arrow be directed from a node m' to its sibling m. The constructed path is the path from the root of the tree to m concatenated with the path constructed recursively in the subtree rooted at m. (If there is no arrow in the subtree rooted at m a path from m to any leaf of the subtree suffices.) Thus, a tree with a fault cannot have a 0-legal arrow diagram. Thus $T(n, 0) \leq 0$.

Next, let $n = b$. If there are 2^n faults, then every leaf is faulty, and it is not possible to draw an arrow diagram with an outgoing arrow on the path from the root to every faulty leaf. Thus, $T(n, n) \leq 2^n - 1$.

Finally, suppose that $0 < b < n$. We show that there is a way of placing $T(n-1, b) + T(n-1, b-1) + 2$ faults at the leaves such that in any legal arrow diagram either there must be at least $b+1$ incoming arrows on some path without any outgoing arrow or there must be a faulty leaf with no outgoing arrow in its path. We place $T(n-1, b) + 1$ worst-case faults in the left subtree and $T(n-1, b-1) + 1$ worst-case faults in the right subtree. Assume that it is possible to place arrows in the tree such that every path to a faulty leaf has an outgoing arrow and every path from the root to a leaf that has no outgoing arrows has at most b incoming arrows. We look at the placement of arrows in the left subtree. Since there are more than $T(n-1, b)$ faults, there must be a path from the root of this subtree to a leaf that has $b+1$ incoming arrows and no outgoing arrows or there must be path from the root of this subtree to a fault with no outgoing arrow. Either of these cases imply that the root of the left subtree must have an arrow from itself to its sibling. Now look at the right subtree. It cannot be the case that there is a path from the root of the right subtree to a faulty leaf with no outgoing arrow, since then there would be no outgoing arrow for the path from the root of T to this fault. Further, no path from the root of the right subtree to a leaf of the right subtree can have more than $b-1$ incoming arrows without having an outgoing arrow, since otherwise there would be a path from the root of the tree to that leaf with more than b incoming arrows without an outgoing arrow. Thus the right subtree must be $(b-1)$-legal. However, the right subtree has more than $T(n-1, b-1)$ worst-case faults. This is a contradiction. □

COROLLARY 2.3.2. *For all $n \geq b \geq 1$, $T(n, b) = O(n^b)$.*

Proof. The recurrence for $T(n, b)$ is bounded from above by the recurrence that we had for $S(n, b)$ in section 2.1 and hence $T(n, b) = O(S(n, b))$. Using Lemma 2.1.1, $T(n, b)$ is $O(n^b)$. □

THEOREM 2.3.3. *For any constants l and d, there is a constant $k = d + (l-1)2^d$ such that there is a way of placing $O(\log^k N)$ faults in the leaves of an N-leaf complete binary tree T such that there is no level-preserving embedding of an N-leaf fault-free complete binary tree T' in T with load l and dilation d.*

Proof. We begin by placing a set of faults of cardinality $O(\log^k N)$ at the leaves of T such that this fault pattern has no k-legal arrow diagram, where $k = d + (l-1)2^d$ and $N = 2^n$. This is possible because $T(n, k) + 1$ is $O(\log^k N)$. Now suppose for the sake of contradiction that there is an embedding of T' into T with load l and dilation d with the property that nodes on level i of T' are mapped to nodes on level i of T and no nodes of T' are mapped to faulty nodes of T. Annotate the tree T with arrows as follows. For any two siblings in the tree, draw an arrow from the sibling whose subtree has a smaller number of leaves of T' mapped to it to the sibling that has a larger number of leaves mapped to it. If the number of leaves mapped to each of the two subtrees is equal then no arrow is drawn.

We now show that the annotated tree is b-legal for some $b \leq k$, which is a contradiction. The path from the root of a tree to any faulty leaf must have an outgoing arrow, since no node of T' is mapped to a faulty leaf. Hence, the first criterion in the definition of a b-legal tree is satisfied. Let b be the maximum number of incoming arrows on a path without an outgoing arrow. We ignore the last d levels of the tree. Therefore, there is a path in T, $m_0, m_1, \ldots, m_{n-d}$, where m_0 is the root and m_i is a node in level i of the tree, that has at least $b-d$ incoming arrows without

any outgoing arrows. Let l_i, $0 \leq i \leq n-d$, denote the average number of leaves of T' that are embedded into each leaf of the subtree of T rooted at node m_i. Clearly, l_0 is 1. If there is no incoming arrow into node m_i, then the split of leaves of T' is even and hence $l_i = l_{i-1}$. Suppose there is an incoming arrow into node m_i from its sibling m'_i. Then $l_i > l_{i-1}$. Further, $l_i \geq l_{i-1} + 2^{-d}$. To see why, consider the subtrees of T' rooted at level $i+d$. The nodes in each of these subtrees can be mapped entirely within either the subtree rooted at m_i or entirely within the subtree rooted at m'_i but never to the nodes in both. To see why, note that if a node in one of these subtrees that was mapped to the subtree rooted at m_i and a neighbor at an adjacent level was mapped to the subtree rooted at m'_i, then the dilation of the edge between them would be more than d. (In fact, the dilation would have to be at least $2d+3$, since the subtrees are separated by a distance of $2d+2$, and any two nodes connected by an edge in T' must be mapped to different levels in T.) Thus, the subtree rooted at m_i must have at least 2^{n-i-d} more leaves of T' mapped to it than the subtree rooted at m'_i. Hence, $l_i \geq l_{i-1} + (2^{n-i-d}/2^{n-i}) = l_{i-1} + 2^{-d}$. Since there are at least $b-d$ incoming arrows on the path, $l_{n-d} \geq 1 + (b-d)2^{-d}$. Note that there is at least one leaf in the subtree rooted at m_{n-d} that has load at least l_{n-d}. Therefore, $l \geq l_{n-d}$. This implies that $b \leq d + (l-1)2^d = k$. But there can be no k-legal arrow placement for the fault pattern chosen for T. This is a contradiction. □

THEOREM 2.3.4. *For any constants l and d, there is a constant $k = d + (l-1)2^d$ such that there is a way of choosing $\Theta(\log^k N)$ faults in an N-input butterfly B such that there is no level-preserving embedding of an N-input butterfly B' in B with load l and dilation d.*

Proof. The proof is similar to that of Theorem 2.3.3. Let B be a butterfly with faults and let B' be the fault-free version of B. We can associate a tree T with B as follows: the root of T represents the entire butterfly B. Its children represent the two subbutterflies of dimension $\log N - 1$ (between levels 1 and $\log N$). Each child is subdivided recursively until each leaf of the tree T represents a distinct node in level $\log N$ of the butterfly B. We choose the same set of worst-case faults in the leaves of T as in Theorem 2.3.3. The faulty nodes of B are the nodes in level $\log N$ of B that correspond to the faulty leaves of T. Given a level-preserving embedding of B' into B with load l and dilation d, we can produce a b-legal placement of arrows in T in a manner similar to the previous proof. Given two siblings m and m', draw an arrow from m' to m if the there are more nodes in level $\log N$ of B' mapped to the subbutterfly of B represented by tree node m than the subbutterfly represented by tree node m'. Let m and m' be on level j of T. As before, due to dilation considerations, the smaller subbutterflies of B' spanning levels $j+d$ to n must be mapped entirely within the subbutterfly of B represented by m or within the subbutterfly represented by m' but never to both. The rest of the proof is similar to Theorem 2.3.3. □

The following theorem is stated for two-dimensional meshes of trees. An analogous theorem can be proved for any constant-dimension mesh of trees.

THEOREM 2.3.5. *For any constants l and d, there is a constant $k = d + (l-1)2^d$ such that there is a way of choosing $\Theta(\log^k N)$ faults in a $\sqrt{N} \times \sqrt{N}$ mesh of trees M such that there is no level-preserving embedding of a $\sqrt{N} \times \sqrt{N}$ mesh of trees M' in M with load l and dilation d.*

Proof. The proof is similar to that of Theorem 2.3.4. Let M be a mesh of trees with faults and let M' be the fault-free version of M. The nodes in level $\log N$ of M and M' are arranged in the form of a two-dimensional $\sqrt{N} \times \sqrt{N}$ mesh. We refer to these nodes as the mesh nodes. We can associate a tree T with the mesh nodes of M

as follows: the root of T represents the entire mesh. Divide the mesh vertically into two equal parts and let each child represent one of the halves. At the next level of the tree divide each of the halves horizontally into two equal parts. Divide alternately, either vertically or horizontally, until reaching individual mesh nodes, which are each represented by a distinct leaf of the tree. We choose the same set of worst-case faults in the leaves of T as in Theorem 2.3.3. The faulty nodes of M are the mesh nodes of M that correspond to the faulty leaves of T. Given a level-preserving embedding of M' into M with load l and dilation d, we can produce a b-legal placement of arrows in T in a manner similar to the previous proofs. Given two siblings m and m', draw an arrow from m' to m if there are more mesh nodes of M' mapped to the submesh of M represented by tree node m than the submesh represented by tree node m'. As before, due to dilation considerations, the smaller submeshes of M' must be mapped entirely within the submesh of B represented by m or within the submesh represented by m' but never to both. The rest of the proof is similar to Theorem 2.3.3. □

3. Fault-tolerant routing. In this section, we present algorithms for routing around faults in hypercubic networks. Section 3.1 presents algorithms for routing packets in a butterfly network with faulty nodes, while section 3.2 presents algorithms for establishing edge-disjoint paths between the inputs and outputs of an $O(1)$-dilated Beneš network with faulty switches.

3.1. Fault-tolerant packet routing. In this section we show how to route packets in an N-input butterfly network with f worst-case faults. In particular, we focus on the problem of routing packets between the nodes of the network in a one-to-one fashion. This type of routing is also called *permutation routing.* (See [37] for references to permutation routing algorithms.) In a permutation routing problem, every node is the origin of at most one packet and the destination of at most one packet. We show that there is some set of $N - 6f$ rows (where $0 \leq f \leq N/6$) such that it is possible to route any permutation between the nodes in these rows in $O(\log N)$ steps using constant-size queues, with high probability. The same result (without the high probability caveat) was previously shown for the expander-based multibutterfly network [28]. A special case of this result is that when $f \leq N/12$ we can route arbitrary permutations between a majority of nodes in the butterfly. Note that this is optimal to within constant factors since N faults on level $(\log N)/2$ partitions the butterfly into many disjoint small connected components.

It will be convenient for us to view the packets as being routed on a larger network with $4 \log N + 1$ levels and N rows. The network consists of four stages. Stage i consists of those nodes in levels $i \log N$ through $(i + 1) \log N$ for $0 \leq i \leq 3$. Note that each pair of consecutive stages shares a level of nodes. The nodes in stages 0 and 3 are connected by straight edges only. Stages 1 and 2 consist of a pair of back-to-back butterflies isomorphic to the Beneš network. In analogy with the Beneš network, the nodes in levels $\log N$, $2 \log N$, and $3 \log N$ are called input nodes, middle nodes, and output nodes, respectively. Note that this larger network can be embedded in a butterfly network so that the jth row of the larger network is mapped to the jth row of the butterfly, and at most one node from each stage of the larger network is mapped to each node of the butterfly. The embedding has load 4, congestion 4, and dilation 1.

We start by describing Valiant's algorithm [54] for permutation routing on a butterfly without faults. Each node in stage 0 is the source of at most one packet, and each node in stage 3 is the destination of at most one packet. (Thus in the underlying butterfly, each node is both the source and destination of at most one packet.) In

stage 0, a packet travels along its row to the input node in that row (say, m). In stage 1, the packet goes from m to a random middle node (say, m''). In stage 2, the packet goes from m'' to the output node m' in the row of its destination. In stage 3, the packet travels along the row of m' until it reaches its destination. Valiant showed that these paths, which have length at most $4 \log N$, also have congestion $O(\log N)$ with high probability. In networks such as the butterfly with $O(\log N)$ levels, as long as the (leveled) paths of the packets have congestion $O(\log N)$, a Ranade-type queuing protocol can be used to route the packets in $O(\log N)$ steps using constant-size queues, with high probability [29]. Therefore, it is sufficient to derive high-probability bounds on the congestion of the paths in a routing scheme.

Our goal is to identify a large set of "good" nodes in a faulty butterfly between which we can route permutations using an algorithm like Valiant's. A node in the four-stage network is faulty if the corresponding node in the underlying butterfly is faulty. Since stages 0 and 3 require a fault-free row, any node in a row with a fault is declared to be *bad*. Furthermore, in stage 1, every packet needs a sufficient number of random choices of middle nodes. For every input node m, let $REACH(m)$ be defined to be the set of middle nodes reachable from m using fault-free paths of length $\log N$ from level $\log N$ to level $2 \log N$. Also, for every output node m', let $REACH(m')$ be the set of middle nodes reachable using fault-free paths from level $3 \log N$ back to level $2 \log N$. Note that if m and m' lie in the same row, then $REACH(m) = REACH(m')$, because the fault pattern in stage 2 is the mirror image of the fault pattern in stage 1. If $|REACH(m)| < 4N/5$ for any input node m, then we declare m and all other nodes in its row to be bad. Any node not declared bad is considered *good*. Note that there are no faults in rows containing good nodes, and every good input or output node can reach at least $4N/5$ middle nodes via fault-free paths. A row in the underlying butterfly is good if the corresponding row in the four-stage network is good.

We now show that only $6f$ rows contain bad nodes. This follows from the fact that only f rows can contain faults and from the fact that $|REACH(m)| \geq 4N/5$ for all but $5f$ input nodes m. The latter fact is proved by setting $t = N/5$ in the following lemma, which will also be used in section 3.2.

LEMMA 3.1.1. *In an N-input butterfly with f worst-case faults, at least $N - fN/t$ input nodes (nodes in level 0) can each reach at least $N - t$ output nodes (nodes in level $\log N$) via fault-free paths of length $\log N$ for any $t \leq N$.*

Proof. For each input node i, let n_i represent the number of output nodes in level $\log N$ that i cannot reach. If the lemma were false, then we would have

$$\sum_{i=0}^{N-1} n_i \geq \left(\frac{fN}{t} + 1 \right) (t + 1).$$

A fault at any level of the butterfly lies on precisely N paths from input nodes to output nodes. Hence $\sum_{0=1}^{N-1} n_i = fN$. Combining the equation with the inequality yields $fN \geq (fN/t + 1)(t + 1)$, which is a contradiction. Hence the lemma must be true. □

In order to route any permutation between the good nodes, we use Valiant's algorithm except that in stage 1, in order to route a packet from input m to output m', we randomly select a middle node m'' from $REACH(m) \cap REACH(m')$. (One way to do this is to store at each input m a table containing information about $REACH(m) \cap REACH(m')$ for each output m'.) Since m and m' are good input and output nodes $|REACH(m) \cap REACH(m')|$ is at least $3N/5$. We now prove that the paths selected in this manner have congestion $O(\log N)$ with high probability.

LEMMA 3.1.2. *For any constant $k > 0$, the randomly selected paths have congestion $O(\log N)$ with probability at least $1 - 1/N^k$. Furthermore, these paths are leveled and have length at most $4 \log N$.*

Proof. The lengths of the paths are clearly at most $4 \log N$ since the paths traverse the butterfly four times, once in each stage. We bound the congestion as follows. Every good node sends and receives one packet. Thus, the congestion of any node in stages 0 and 3 is trivially at most $(\log N + 1)$. Consider a node s in level l of the butterfly in stage 1. There are $2^l(\log N + 1)$ packets that could pass through this node. A packet passes through this node if and only if it selects as a random middle node one of the $2^{\log N - l}$ middle nodes reachable from this node. Note that the set of possible choices of middle nodes for any input node m and output node m' is $REACH(m) \cap REACH(m')$. Since both m and m' are good nodes, $|REACH(m)| \geq 4N/5$ and $|REACH(m')| \geq 4N/5$. This implies that $|REACH(m) \cap REACH(m')| \geq 3N/5$. Thus the probability that the packet chooses a middle node that is reachable from s is at most $2^{\log N - l}/(3N/5)$. Therefore, the expected number of packets that passes through a node in stage 1 is at most $(2^l(\log N + 1))(2^{\log N - l})/(3N/5) = 5(\log N + 1)/3$. We can use Chernoff-type bounds [46] to show that the number of packets that pass through s in stage 1 is $O(\log N)$ with probability at least $1 - 1/2N^k$ for any constant k. The calculation for a node in stage 2 is exactly analogous. Thus the congestion is $O(\log N)$ with probability at least $1 - 1/N^k$. □

The following theorem summarizes the result presented in this section.

THEOREM 3.1.3. *In an N-input butterfly network with f worst-case faults, where $0 \leq f \leq N/6$, there is some set of $N - 6f$ "good" rows whose nodes can serve as the origins and destinations of any permutation routing problem. Furthermore, there is a routing algorithm such that, for any constant $k > 0$, there is a constant $C > 0$ such that the algorithm routes all the packets in any permutation (using routing tables) in at most in $C \log N$ steps using constant-size queues, with probability at least $1 - 1/N^k$.*

Proof. Lemma 3.1.1 shows how to identify the $N - 6f$ rows that are to serve as the sources and destinations of the packets. The lemma is applied for the case $t = N/5$. To route from an input node m to an output node m', a packet must select a random intermediate destination m'' that can be reached from both m and m'. The choice is made by consulting a table of all such m''. Lemma 3.1.2 shows that, with high probability, these paths have congestion $O(\log N)$. Finally, once the paths are selected, the algorithm for routing on leveled networks [29] can be applied to deliver the packets in $O(\log N)$ steps, with high probability using constant-size queues. □

3.1.1. Packet routing without routing tables. In the previous algorithm, every good input node m was required to store a table containing information about $REACH(m) \cap REACH(m')$ for every good output node m'. In this section, we show that is possible to route packets in a faulty butterfly without using such routing tables. The information about the placement of the faults is used only during the reconfiguration when the good and bad nodes are identified. This information is not needed for the routing itself. We assume that any packet that attempts to go through a fault is simply lost. We further assume that a node that receives a packet sends back an acknowledgement message (ACK) to the sender. Each ACK message follows the path of the corresponding packet in reverse. An ACK is generally smaller than a message and requires at most $O(\log N)$ bits to specify its path and the identity of the message that it is acknowledging. The algorithm for routing proceeds in rounds. There are a total of $(A \log \log N) + 1$ rounds (for some constant A). Each round consists of the following steps (R takes values from 0 to $A \log \log N$ and denotes the round number).

SEND-PACKET: In stage 0, if packet p has not yet been delivered to its destination, send 2^R identical copies of p to the input node m in its row. In stage 1, send each copy of the packet independently to a random middle node. In stage 2, send each copy to the appropriate output node m'. In stage 3, send each copy to the appropriate destination node in that row.

RECEIVE-PACKET: If a packet is received send an ACK along the same path that the packet came through but in reverse.

WAIT: Wait $B \log N$ steps before starting the next round.

THEOREM 3.1.4. *For any constant $k > 0$, there are constants A and B such that the algorithm routes any permutation on the nodes of the $N - 6f$ good rows of an N-input butterfly with f faults in $O(\log N \log\log N)$ steps using constant-size queues, with probability at least $1 - \frac{1}{N^k}$, without using routing tables.*

Proof. First we show that there is very little probability that a packet survives $A \log\log N$ rounds without reaching its destination, where A is an appropriately chosen constant. A packet can never encounter a fault in stages 0 and 3 since its source and destination rows are fault-free. Let m and m' denote the input and output nodes that the packet passes through, respectively. In stage 1, if the packet chooses any middle node in $REACH(m) \cap REACH(m')$ then it succeeds in reaching its destination. Since $|REACH(m) \cap REACH(m')| \geq 3N/5$ the probability of this happening is at least $3/5$. Suppose the packet did not get through after $A \log\log N$ rounds. Then $2^{A \log\log N} = \log^A N$ copies of the packet are transmitted in the last round. Note that the probability of each copy surviving is independent of the others. Hence the probability that none of these copies reach their destination is at most $(1-3/5)^{\log^A N}$, which is at most $1/N^{k+3}$, for any constant $k > 0$ and for an appropriate choice of the constant A. Thus, the probability that some packet does not reach its destination is at most $N \log N/N^{k+3}$, which is less than $1/N^{k+1}$.

Next we show that each round takes $O(\log N)$ time with high probability. We assume inductively that at the beginning of round i the total number of packets (counting each copy once) to be transmitted from any row in stage 0 of the algorithm or received by any row in stage 3 of the algorithm is at most $q \log N$ for some constant $q > 1$. Clearly the basis of the induction is true at the beginning of the first round since there are exactly $\log N$ packets sent by each row in stage 0 and received by each row in stage 3. The expected number of copies that are sent from a row in stage 0 or that are destined for a row in stage 4 that do not get through is at most $2q \log N/5$. The value of q is chosen such that the probability that more than $q \log N/2$ copies do not get through in any row can be shown to be small, i.e., at most $1/AN^{k+2}$, for any constants A and k, using Chernoff-type bounds. At the beginning of the next stage, each unsent copy is duplicated and hence, with high probability, the number of packets in any row in the next round is at most $q \log N$. Since there are $(A \log\log N)+1$ rounds, the probability that the inductive hypothesis does not hold in the beginning of any one round is at most $((A \log\log N) + 1)/AN^{k+2}$, which is less than $1/N^{k+1}$.

Now we assume that the inductive hypothesis is true and show that each round takes only $B \log N$ steps, for some constant B, with high probability. Consider any round i. From the inductive hypothesis, the congestion of any node in stage 0 or 3 is at most $q \log N$ which is $O(\log N)$. In stage 1, a node at level l can receive packets from any one of 2^l input nodes, each packet with probability 2^{-l}. The total number of packets that pass through an input node is at most $q \log N$ by our inductive hypothesis. Therefore, the expected number of packets that pass through a node at level l is $q \log N 2^l 2^{-l} = q \log N$. The value of q is chosen so that the probability that

any node gets more than $2q \log N$ packets can be shown to be at most $1/AN^{k+2}$, using Chernoff-type bounds. The analysis for stage 2 is similar. Thus we have shown that if the inductive hypothesis is true, the congestion of any node is $O(\log N)$ with high probability. Therefore, using the algorithm for routing on leveled networks [29] to schedule the packets, the routing completes in $C \log N$ steps with probability at least $1-2/AN^{k+2}$ for an appropriate constant C. The ACKs follow the paths of the packets in the reverse direction. Therefore, the congestion in any node due to ACKs can be no larger than the congestion due to packets and is also therefore $O(\log N)$. Since we are using the algorithm for routing on leveled networks to schedule the packets, the probability that some ACK does not reach its destination in $D \log N$ time is also at most $2/AN^{k+2}$ for some suitably large constant D. We choose the constant B in the algorithm to be at least $C + D$ so that the algorithm waits long enough for both the packet routing and the routing of ACKs to finish before starting the next round of routing. The probability that either the packet routing or the ACK routing fails to complete in some round is at most $4A \log \log N/AN^{k+2}$, which is less than $1/N^{k+1}$.

The probability that either some packet remains untransmitted after the last round or that the inductive hypothesis does not hold for some round or that some round fails to complete in $B \log N$ steps is at most $3/N^{k+1}$, which is less than $1/N^k$. Thus, the algorithm successfully routes every packet to its destination in $O(\log N \log \log N)$ steps with probability at least $1 - 1/N^k$. □

If the number of worst-case faults is smaller then there is a simpler way of routing in $O(\log N)$ steps without using routing tables or creating duplicate packets.

THEOREM 3.1.5. *Given an N-input butterfly with $N^{1-\epsilon}$ worst-case faults (for any constant $\epsilon > 0$), it is possible to identify $N - o(N)$ good rows in the butterfly such that any permutation routing problem on the good nodes can be routed in $O(\log N)$ steps using constant-size queues with probability greater than $1 - 1/N^k$, for any fixed constant k, without using routing tables.*

Proof. We define the "good" nodes in the butterfly as follows. A row is good if it contains no faults and the input in that row can reach all but $N^{1-\epsilon/2}$ middle nodes. (Previously a good input was required to reach all but $N/5$ middle nodes.) Using Lemma 3.1.1 with $f = N^{1-\epsilon}$ and $t = N^{1-\epsilon/2}$, we see that the number of good rows is least $N - N^{1-\epsilon/2}$. The algorithm is the same as the previous algorithm except that we don't create any duplicate packets and we now need only a constant number of routing rounds with high probability. This is because each unsent packet at each round has probability at most $\Theta(1/N^{\epsilon/2})$ of hitting a fault. Therefore, it is sufficient to have $\Theta(k/\epsilon)$ rounds (a constant) before every packet is delivered, with probability at least $1 - 1/N^k$. □

3.2. Fault-tolerant circuit switching. In this section, we examine the ability of the Beneš network to establish disjoint paths between its inputs and outputs when some of its switches fail. We assume that no path can pass through a faulty switch. The main result of this section is a proof that for arbitrarily small positive constants ϵ and δ, there is a constant b such that given a b-dilated $(\log N)$-dimensional Beneš network with $f = N^{1-\epsilon}$ worst-case switch failures, we can identify a set of $N - 4N^{1-\delta}$ input and output switches such that it is possible to route edge-disjoint paths in any permutation between the corresponding input and output edges. (A *b-dilated* Beneš network is one in which each edge is replaced by b parallel edges, and each 2×2 switch is replaced by a $2b \times 2b$ switch.) At each input switch, two of the b incoming edges are used as inputs, and at each output switch, two of the b outgoing edges are used as outputs.

In a $(\log N)$-dimensional Beneš network, levels 1 through $2\log N-1$ can be decomposed into two disjoint sub-Beneš networks of dimension $\log N - 1$, a top sub-Beneš network, and a bottom sub-Beneš network. Note that the two paths that originate from input edges that share an input switch cannot use the same sub-Beneš network. The same is true for paths that end on output edges that share the same output switch. A full permutation consists of a set of $2N$ input–output pairs to be connected by edge-disjoint paths. The standard algorithm for setting the switches in a Beneš network, due to Waksman [55], uses bipartite graph matching to split the set of $2N$ pairs into two sets of N pairs which are then each routed recursively in one of the smaller sub-Beneš networks.

We now present Waksman's algorithm with a twist. We call this algorithm RANDSET (for RANDom switch SETting). The way RANDSET differs from Waksman's algorithm is that it randomly chooses which of the two sets of N pairs to route through the top (and bottom) sub-Beneš network. The input to RANDSET is a permutation ϕ represented as a $2N \times 2N$ bipartite graph. The nodes of the graph represent the $2N$ input edges and the $2N$ output edges of the network. An edge in the bipartite graph from input i to output $\phi(i)$ indicates that a path must be routed from i to $\phi(i)$ in the network. The first step is to merge pairs of nodes in the bipartite graph that correspond to input edges (or output edges) that share the same input switch (or output switch). The result is a 2-regular $N \times N$ bipartite graph. The second step is to split the edges of this graph into two perfect matchings, M_0 and M_1. (See [42] for a nice proof that such a split is possible.) Next, we pick a binary value for random variable X at random. If $X = 0$ then we recursively route the paths in matching M_0 through the top sub-Beneš network and those in M_1 through the bottom sub-Beneš network. If $X = 1$ we do the opposite. The following lemma shows that RANDSET chooses the path from i to $\phi(i)$ uniformly from among all possible paths.

LEMMA 3.2.1. *For any i, the path chosen by algorithm RANDSET between input i and output $\phi(i)$ in a $2N$-input Beneš network passes through any of the N middle switches (switches in level $\log N$) with equal probability $(1/N)$.*

Proof. At the first stage, the path from i to $\phi(i)$ goes to the top or the bottom sub-Beneš network with probability $1/2$ depending on whether the matching that contains the edge corresponding to this input–output pair is chosen to be routed through the top or the bottom. The decisions made at the succeeding levels of the recursion are similar and independent of all other decisions. □

It is important to remember that given a permutation, the *paths themselves could be highly correlated* and determining one path gives some information about the others.

We classify the input and output switches of the Beneš network as either good or bad depending on whether they can reach a sufficiently large number of middle switches. In a fault-free Beneš network, there is a path from each input (and output) switch to each of the N middle switches. The middle switches in fact form the leaves of a complete binary tree with the input (or output) switch as the root. The faults could make some of these paths unusable. We declare an input (or output) switch *bad* if the number of middle switches that it cannot reach exceeds a certain threshold. The threshold is chosen so that it is possible to establish edge-disjoint paths between the *good* (i.e., not bad) inputs and outputs in any permutation. (The two inputs coming into an input (or output) switch are good or bad depending on whether the corresponding input (or output) switch is good or bad.)

Let $BAD(t)$ be the set of input and output switches for which more than t middle switches are unreachable. The first and last $\log N + 1$ levels of the Beneš network each

form a $\log N$-dimensional butterfly. Applying Lemma 3.1.1 to each of these butterflies separately, we know that $|BAD(t)| < 2fN/t$. (Note that these butterflies share the middle level of switches and hence might share some faults.)

THEOREM 3.2.2. *For any constants $0 < \delta < \epsilon \leq 1$, there exists a constant $b = \lceil 1 + (2-\epsilon)/(\epsilon - \delta) \rceil$ such that a $2N$-input b-dilated Beneš network with $N^{1-\epsilon}$ worst-case switch faults has a set of $N - 4N^{1-\delta}$ input switches and output switches between whose input and output edges it is possible to route any permutation using edge-disjoint paths.*

Proof. We declare any input or output switch in $BAD(N^{1+\delta-\epsilon}/2)$ to be bad. Since we need the number of good input switches and good output switches to be equal we may have to declare some extra input switches or output switches to be bad. From Lemma 3.1.1, we know that $|BAD(N^{1+\delta-\epsilon}/2)| \leq 4N^{1-\delta}$. Thus, the number of good input switches (or output switches) is at least $N - 4N^{1-\delta}$.

We now prove that we can route any permutation between the good inputs and good outputs using edge-disjoint paths. In this proof, we simply show that for every permutation such a set of paths *exists*, without showing how to compute these paths efficiently. Later, we give an efficient procedure for computing these paths.

Given a permutation ϕ on the good inputs and outputs, we select paths using RANDSET in b rounds. In the first round, we route all the paths using RANDSET. Some of these paths pass through faults in the network. The number of paths that pass through faults is at most $2N^{1-\epsilon}$, since each fault appears on at most two paths. These paths are not permissible and must be rerouted in the second round using RANDSET. Note that every good input switch (or output switch) has at most $N^{1+\delta-\epsilon}/2$ unreachable middle switches. Thus, from Lemma 3.2.1, the probability that any one of the paths hits a fault in the first $\log N + 1$ levels is at most $N^{-(\epsilon-\delta)}/2$. The probability that it hits a fault in the second $\log N + 1$ levels is also at most $N^{-(\epsilon-\delta)}/2$. The net probability that the path hits a fault is at most $N^{-(\epsilon-\delta)}$. Even though the probabilities that any two paths hit a fault is correlated, the expected number of paths that hit faults in the second round is at most $2N^{1-\epsilon}N^{-(\epsilon-\delta)}$. This implies that with nonzero probability RANDSET finds a set of paths such that at most $2N^{1-\epsilon-(\epsilon-\delta)}$ paths hit faults. Note that this also means that there *exists* a way of selecting the paths so that at most $2N^{1-\epsilon-(\epsilon-\delta)}$ paths hit faults. We select paths such that this criterion is satisfied and route the paths that hit faults again using RANDSET. We continue to do the rerouting until the expected number of paths that hit faults drops below 1. At this point, with nonzero probability RANDSET routes all of the paths without hitting any faults. In particular, such a set of paths exists. The expected number of paths that hit faults in the ith round is $2N^{1-\epsilon-(i-1)(\epsilon-\delta)}$. Thus, for $b = \lceil 1 + (2-\epsilon)/(\epsilon-\delta) \rceil$ the number of paths that hit faults at the end of the bth round is less than 1. Therefore, all paths are routed by the end of the bth round. Since we use at most b rounds of routing and since each edge of the Beneš network has been replaced by b edges, we obtain edge-disjoint paths for the permutation. □

3.2.1. Derandomizing RANDSET. In the proof of Theorem 3.2.2, we show the existence of edge-disjoint paths by using the fact that algorithm RANDSET finds them with nonzero probability. In this section we construct a deterministic algorithm that finds these paths using the technique due to Raghavan [45] and Spencer [50] to remove the randomness. Like Waksman's algorithm for finding the switch settings in a fault-free Beneš network with N input switches, the algorithm runs in $O(N \log N)$ time.

Let P be the random variable that denotes the number of paths that go through

faults at some round of rerouting. Let X be the binary random variable used by RANDSET to make its random decision to select which matching is to be routed through which sub-Beneš network. Let us further define two random variables P_l and P_r to denote the number of paths that RANDSET routes through faults in the left butterfly and the right butterfly, respectively. Let $U(P)$ be an upper bound on $E(P)$ that is defined as $U(P) = E(P_l) + E(P_r)$. In the proof of Theorem 3.2.2, we used the fact that with nonzero probability RANDSET finds a set of paths in which at most $U(P)$ paths hit faults. We define an algorithm DSET (for deterministic switch SETting) that deterministically finds such a set of paths. Algorithm DSET is the same as RANDSET except that instead of selecting a random value for X, we select the "better" choice for X as follows. We compute $U(P|(X = i)) = E(P_l|(X = i)) + E(P_r|(X = i))$ for $i = \{0, 1\}$. We then choose X to be the value of i that yields the minimum of the two values computed above.

THEOREM 3.2.3. *Given a (partial) permutation ϕ to be routed, algorithm DSET deterministically computes paths such that at most $U(P)$ paths hit faults, and DSET has the same asymptotic running time as RANDSET.*

Proof. We prove the theorem by induction on the size of the Beneš network. The base case is trivial. Now consider a $2N$-input Beneš network with a (partial) permutation ϕ to route. Let P_f and P_e denote the number of paths that hit faults in the first and last levels of the Beneš network, respectively. Also, let P_t and P_b denote the number of paths that hit faults in the top and bottom sub-Beneš networks (but not in the first or last levels of the Beneš network), let $P_{t,l}$ and $P_{t,r}$ denote the number of paths that hit faults in the left and right halves of the top sub-Beneš network, and let $P_{b,l}$ and $P_{b,r}$ denote the number of paths that hit faults in the left and right halves of the bottom sub-Beneš network. Finally, let i be the value chosen for X in step 2 of DSET. The total number of paths that hit faults P is bounded as follows:

$$P \le P_f + U(P_t|(X = i)) + U(P_b|(X = i)) + P_e \tag{3.1}$$

$$= P_f + E(P_{t,l}|(X = i)) + E(P_{t,r}|(X = i)) + E(P_{b,l}|(X = i)) + E(P_{b,r}|(X = i)) + P_e \tag{3.2}$$

$$= E(P_l|(X = i)) + E(P_r|(X = i)) \tag{3.3}$$

$$= U(P|(X = i)) \tag{3.4}$$

$$\le U(P). \tag{3.5}$$

These inequalities have the following explanation. Independent of the choice of i, P_f paths are blocked at the first level and P_e are blocked at the last. Once i is chosen, we know by induction that at most $U(P_t|(X = i))$ paths are blocked in the top sub-Beneš network and at most $U(P_b|(X = i))$ are blocked in the bottom sub-Beneš network. Hence inequality (3.1) holds. Equation (3.2) is derived by substituting the definitions of $U(P_t|(X = i))$ and $U(P_b|(X = i))$. Equation (3.3) is derived by observing that

$$E(P_l|(X = i)) = P_f + E(P_{t,l}|(X = i)) + E(P_{b,l}|(X = i))$$

and

$$E(P_r|(X = i)) = P_e + E(P_{t,r}|(X = i)) + E(P_{b,r}|(X = i)).$$

Equation (3.4) follows from the definition of $U(P|(X = i))$. Finally, (3.5) holds by the choice of i.

Now we deal with the question of how fast $U(P|(X = i))$ can be calculated in step 2 of DSET for $i \in \{0, 1\}$. For every switch m in the Beneš network, let $REACH(m)$ be the set of middle switches reachable from switch m using fault-free paths. We can precompute $|REACH(m)|$ as follows. The value for the middle switches are trivially known. We then compute the values for levels on both sides adjacent to levels where the values are known and continue in this manner. This takes only $O(N \log N)$ steps of precomputation and does not affect the asymptotic time complexity of the algorithm. Given the values of $|REACH(m)|$, the values of $U(P|(X = i))$ can be easily calculated by summing up the appropriate values of $|REACH(m)|$. This is an $O(N)$ time computation. Since step 1 of the algorithm takes N time just to set N switches in the first level, this does not affect the asymptotic time complexity. Hence using DSET yields the same asymptotic time complexity as RANDSET and takes time linear in the size of the Beneš network. □

4. Emulations on faulty butterflies. In this section, we show that for any constant $\epsilon > 0$, a $(\log N)$-dimensional butterfly with $N^{1-\epsilon}$ worst-case faults (the host H) can emulate any computation of a fault-free $(\log N)$-dimensional butterfly (the guest G) with only constant slowdown. We assume that a faulty node cannot perform computations and that packets cannot route through faulty nodes. For simplicity we assume that both the guest and host butterflies wrap around; i.e., the nodes of level 0 are identified with the nodes of level $\log N$.

We model the emulation of G by H as a pebbling process. There are two kinds of pebbles. With every node v of G and every time step t, we associate a *state pebble* (s-pebble) $\langle v, t\rangle$ that contains the entire state of the computation performed at node v at time t. The s-pebble contains local memory values, registers, stacks, and anything else that is required to continue the computation at v. We view G as a directed graph by replacing each undirected edge between nodes u and v by two directed edges: one from u to v and the other from v to u. With each directed edge e and every time step t, we associate a *communication pebble* (c-pebble) $[e, t]$ that contains the message transmitted along edge e at time step t.

The host H emulates each step t of G by creating an s-pebble $\langle v, t\rangle$ for each node v of G and a c-pebble $[e, t]$ for each edge e of G. A node of H can create an s-pebble $\langle v, t\rangle$ only if contains s-pebble $\langle v, t-1\rangle$ and all of the c-pebbles $[e, t-1]$, where e is an edge into v. It can create a c-pebble $[g, t]$ for an edge g out of v only if it contains an s-pebble $\langle v, t\rangle$. A node of H can also transmit a c-pebble to a neighboring node of H in unit time. A node of H is not permitted to transmit an s-pebble since an s-pebble may contain a lot of information. Note that H can create more than one copy of an s-pebble or c-pebble. The ability of H to create redundant pebbles is crucial to our emulation schemes. In our emulations, each node of H is assigned a fixed set of nodes of G to emulate and creates s-pebbles for them for each time step.

4.1. Assignment of nodes of G to nodes of H. We now show how to map the computation of G to the faulty butterfly H. The host H has $N^{1-\epsilon}$ arbitrarily distributed faults. We first divide H into subbutterflies of dimension $(\epsilon \log N)/2$ spanning levels $(i\epsilon \log N)/2$ through $((i + 1)\epsilon \log N)/2$ for integer $i = 0$ to $2/\epsilon - 1$. (Without loss of generality, we assume that $2/\epsilon$ and $\epsilon \log N/4$ are integers.) Note that every input node of a subbutterfly is also an output node of another subbutterfly and vice versa. Each band of $((\epsilon \log N)/2) + 1$ levels consists of $N^{1-\epsilon/2}$ disjoint subbutterflies. Thus, there are a total of $(2/\epsilon)N^{1-\epsilon/2}$ subbutterflies. The faults in the network may make some of these subbutterflies unusable. We identify "good" and "bad" subbutterflies according to the following rules.

Rule 1. A subbutterfly that contains a node that lies in a butterfly row in which there is a fault is a *bad* subbutterfly (even if the fault lies outside of the subbutterfly).

Rule 2. In order to apply Rule 2, we embed a Beneš network in the butterfly. The edges of the first stage of the Beneš network traverse the butterfly in increasing order of dimension and the edges of the second stage in decreasing order of dimension. The input switches, the middle switches, and the output switches of the Beneš network are all embedded in level 0 of the butterfly (which is the same as level $\log N$). For $\delta = 2\epsilon/3$, identify the set of bad inputs/outputs (they are the same set here) according to the procedure outlined in the proof of Theorem 3.2.2 in section 3.2. Any subbutterfly that contains a node that has a bad input/output at the end of its butterfly row is a bad subbutterfly.

LEMMA 4.1.1. *For any $\epsilon > 0$, the number of rows in which there is either a fault or a bad input or output is at most $N^{1-\epsilon} + 4N^{1-2\epsilon/3}$.*

Proof. The number of rows containing a fault is at most $N^{1-\epsilon}$, since there are at most $N^{1-\epsilon}$ faults. By Theorem 3.2.2, for $\delta = 2\epsilon/3$, the number of bad inputs and outputs (they are the same nodes) is at most $4N^{1-\delta} = 4N^{1-2\epsilon/3}$. □

LEMMA 4.1.2. *For any $\epsilon > 0$, at least half the subbutterflies of H are good for sufficiently large N.*

Proof. The total number of subbutterflies is $(2/\epsilon)N^{1-\epsilon/2}$. By Lemma 4.1.1, the number of rows containing either a fault or a bad input or output is at most $N^{1-\epsilon} + 4N^{1-2\epsilon/3}$. Since each bad row passes through $2/\epsilon$ different subbutterflies, the total number of subbutterflies identified as bad by Rules 1 and 2 cannot exceed $2(N^{1-\epsilon}+4N^{1-2\epsilon/3})/\epsilon$. Observe that $2(N^{1-\epsilon}+4N^{1-2\epsilon/3})/\epsilon \leq N^{1-\epsilon/2}/\epsilon$ for sufficiently large N. □

Now we divide the guest G into overlapping subbutterflies of dimension $(\epsilon \log N)/2$ and map them to the good subbutterflies of H. For any node v of G, at most three nodes of H receive the initial state of the computation of node v, i.e., s-pebble $\langle v, 0\rangle$. (A node v of G can appear as an input in one subbutterfly, an output in another, and a middle node in a third.) These nodes in H create the s-pebbles for v. The mapping proceeds as follows. Take the guest G and cut it into subbutterflies at levels $i\epsilon \log N/2$ for integers $i = 0$ to $2/\epsilon - 1$. Map each subbutterfly to a good subbutterfly of H so that each subbutterfly of H gets at most two subbutterflies of G. Now cut G again, this time at levels $(\epsilon/4 + i\epsilon/2)\log N$ for integers $i = 0$ to $2/\epsilon - 1$. Map the subbutterflies to good subbutterflies of H as before. At most eight nodes of G are mapped to each node of H.

4.2. Building constant congestion paths. We call the nodes v of G belonging to level $i\epsilon \log N/4$, for $i = 0$ to $4/\epsilon - 1$, *boundary nodes* since they lie on the boundary of some subbutterfly of G. Let the set of boundary nodes of G be denoted by $\mathcal{B}_\mathcal{G}$. Similarly we define the nodes in the levels where H was cut to form subbutterflies, i.e., level $i\epsilon \log N/2$ for $i = 0$ to $2/\epsilon - 1$, the boundary nodes of H. Let us denote this set $\mathcal{B}_\mathcal{H}$.

Let ϕ be the function that maps an s-pebble $\langle v, t\rangle$ to the node in H that creates it. The creation of $\langle v, t\rangle$ requires that node $\phi(\langle v, t\rangle)$ of H gets all the c-pebbles $[e, t]$ from some other node of H for every edge e into v. Suppose that $\langle v, t\rangle$ is mapped to some node $m = \phi(\langle v, t\rangle)$ in the interior (i.e., not on the boundary) of a good subbutterfly of H. Then the neighbors of m in H also create the s-pebbles of the neighbors of v in G. In this case m receives the required c-pebbles from its neighbors in H.

On the other hand, if the s-pebble for v is mapped to some node $m \in \mathcal{B}_\mathcal{H}$, the neighbors of m in H may not create the s-pebbles of the neighbors of v in G. However,

since every node v of G is mapped to at least two nodes of H, there is another node m' of H that also creates an s-pebble for v. In particular, *m' is necessarily a node in the center of a subbutterfly of H*, i.e., in level $\epsilon \log N/4$ of the subbutterfly. Node m' of H forwards a copy of the c-pebble $[e, t]$ to node m for each of the edges e into v.

To facilitate the transmission of c-pebbles, we use the results of section 3.2 to establish constant-congestion fault-free paths in H between all pairs of nodes m and m' of H that create the s-pebbles for the same node v in $\mathcal{B}_{\mathcal{G}}$. The number of paths originating in a row of H is simply the number of nodes mapped to subbutterfly boundaries in that row, which is at most $8 \cdot 2/\epsilon = 16/\epsilon$, i.e., a constant. Similarly the number of paths ending in any row is $16/\epsilon$. We can divide the paths into $8/\epsilon$ sets such that each set has at most two paths originating in a row and two paths ending in a row. Note that all paths start and end in rows that have good inputs and outputs for doing Beneš-type routing. Therefore, each set can be routed with dilation $4 \log N$ and with congestion $O(1)$ using the results of section 3.2. Since there are only a constant number of such sets the total congestion is also a constant.

4.3. The emulation. We now formally describe the emulation and prove its properties. Initially, nodes of H contain s-pebbles $\langle v, 0\rangle$ for nodes v of G. We say that H has emulated T steps of the computation of G if and only if for every node v of G, an s-pebble $\langle v, T\rangle$ has been created somewhere in H. The emulation algorithm is executed by every node m of H and proceeds as a sequence of macrosteps. Each macrostep consists of the following four substeps.

1. *Computation step.* For each node v of G that has been assigned to m, if m contains an s-pebble $\langle v, t-1\rangle$ and c-pebbles $[e, t-1]$ for every edge e into v and m has not already created an s-pebble $\langle v, t\rangle$, then it does so.
2. *Communication step.* For each node v whose s-pebble was updated from $\langle v, t-1\rangle$ to $\langle v, t\rangle$ in the computation step and for each edge g from v to a neighbor u of v, node m sends a c-pebble $[g, t]$ to its neighbor in H that creates s-pebbles for u (if such a neighbor exists).
3. *Routing step.* If m has any c-pebble that was created on the previous copy step or that was received on the previous routing step but whose final destination is not m, then m forwards the c-pebble to the next node on the pebble's path to its destination.
4. *Copy step.* If m is a node in the center level (level $\epsilon \log N/4$ in the subbutterfly) and in the communication step m received a c-pebble $[e, t]$ for an edge e into a node v that has been assigned to m, then m makes two copies of the c-pebble, one for each of the two nodes m' and m'' that also create s-pebbles for v. On the next routing step, m forwards each copy to the next node on the path from m to m' or m''.

LEMMA 4.3.1. *Each macrostep takes only a constant number of time steps to execute.*

Proof. At most eight s-pebbles mapped are to each node. Therefore, the computation step takes constant time. Every s-pebble that is updated can cause at most four c-pebbles to be sent in the communication step. Therefore, the communication step takes only constant time. Since only a constant number of paths passes through any node, only a constant number of c-pebbles enters a particular path at any macrostep, and every c-pebble that has not yet reached its destination moves in every macrostep, the number of c-pebbles entering any node during any macrostep is at most a constant. Thus, the routing step and the copy step both take only a constant number of time steps. □

THEOREM 4.3.2. *Any computation on a fault-free butterfly G that takes time T can be emulated in time $O(T + \log N)$ by H.*

Proof. We show that only $O(T + \log N)$ macrosteps are required to emulate a T-step computation of G. The final result then follows from Lemma 4.3.1.

The *dependency tree* of an s-pebble represents the functional dependency of this s-pebble on other s-pebbles and can be defined recursively as follows. As the base case, if $t = 0$, the dependency tree of $\langle v, t\rangle$ is a single node $\langle v, 0\rangle$. If $t > 0$, the creation of s-pebble $\langle v, t\rangle$ requires s-pebble $\langle v, t-1\rangle$ and a c-pebble $[e, t-1]$ for each edge e into node v in G. Each c-pebble is sent by an s-pebble $\langle u, t-1\rangle$ where u is a neighbor of v in G. The dependency tree of $\langle v, t\rangle$ is defined recursively as follows. The root of the tree is $\langle v, t\rangle$ and the subtrees of the root are the dependency trees of $\langle v, t-1\rangle$ and all s-pebbles $\langle u, t-1\rangle$.

Let the emulation of T steps of G take T' macrosteps on H. Let $\langle v, T\rangle$ be an s-pebble that was updated in the last macrostep. We now look at the dependency tree of $\langle v, T\rangle$. We choose a *critical path*, $s_T, s_{T-1}, \ldots, s_0$, of tree nodes from the root to the leaves of the tree as follows. The first node on the path s_T is $\langle v, T\rangle$. Let ϕ be the function that maps an s-pebble $\langle v, t\rangle$ to the node in H that creates it. The creation of s_T requires the s-pebble $\langle v, T-1\rangle$ and c-pebbles $[e, T-1]$. If the s-pebble $\langle v, T-1\rangle$ was created after all the c-pebbles were received then choose s_{T-1} to be $\langle v, T-1\rangle$. Otherwise, choose the s-pebble that sent the c-pebble that arrived last at node $\phi(\langle v, T\rangle)$. After choosing s_{T-1}, we choose the rest of the sequence recursively in the subtree with s_{T-1} as the root. The last s-pebble on the path s_0 is one that was present initially, i.e., at time step 0. We define a quantity l_i as follows. If $\phi(s_i)$ and $\phi(s_{i-1})$ are the same node or neighbors in H, then $l_i = 1$. Otherwise, l_i is the length of the path by which a c-pebble generated by s_{i-1} is sent to s_i. For every tree node s, we can associate a time (in macrosteps) $\tau(s)$ when that s-pebble was created. From the definition of our critical path and because a c-pebble moves once in every macrostep, $\tau(s_i) - \tau(s_{i-1}) = l_i$. Thus,

$$T' = \sum_{0<i\le T} (\tau(s_i) - \tau(s_{i-1})) = \sum_{0<i\le T} l_i.$$

Now suppose that some l_i is greater than one. This corresponds to a long path taken by some c-pebble to go from $\phi(s_{i-1})$ in the center level of a subbutterfly of H to $\phi(s_i)$ in $\mathcal{B}_{\mathcal{H}}$. Thus l_i is the length of one of the constant congestion paths and is at most $(4 \log N)+1$. (In fact, the paths are somewhat shorter, but $(4 \log N)+1$ is a convenient quantity to work with.) The key observation is that since $\phi(s_{i-1})$ is a node in the center level, working down the tree from s_{i-1} there can be no long paths until we reach an s-pebble mapped to the boundary $\mathcal{B}_{\mathcal{H}}$; i.e., $l_{i-j} = 1$ for $1 \le j \le (\epsilon \log N)/4 - 1$. Thus, the extra path length of $4 \log N$ can be amortized over $(\epsilon \log N)/4$ s-pebbles. Hence, $T' = \sum_{0<i\le T} l_i \le (16/\epsilon + 1)T + 4 \log N + 1 = O(T + \log N)$. □

We can extend these results to the shuffle-exchange network using Schwabe's proof [26, 47] that an N-node butterfly can emulate an N-node shuffle-exchange network with constant slowdown, and vice versa.

THEOREM 4.3.3. *Any computation on a fault-free N-node shuffle-exchange network G that takes time T can be emulated in $O(T + \log N)$ time by an N-node shuffle-exchange network H with $N^{1-\epsilon}$ worst-case faults for any constant $\epsilon > 0$.*

Proof. Schwabe [26, 47] shows how to emulate any computation of a butterfly on a shuffle-exchange network with constant slowdown and vice versa. First we use Schwabe's result to map the computation of a butterfly B to the faulty shuffle-exchange network H. Any node of B that is mapped to a faulty node of H is declared

faulty. If there are any routing paths required by the emulation that pass through a faulty node of H, we declare the nodes of B that use this path to be faulty. The faulty nodes of B do no computation, and H is not required to emulate them. Hence, the faulty shuffle-exchange network H can emulate the faulty butterfly network B with constant slowdown. The number of faults in B is only a constant factor larger than $N^{1-\epsilon}$, since both the load and the congestion of the paths used in Schwabe's emulation are constant. Now we use Theorem 4.3.2 to emulate a fault-free butterfly B' on B. Finally, use Schwabe's result (in the other direction) to emulate the fault-free shuffle-exchange network G on the fault-free butterfly B'. Each of these emulations has constant slowdown. Therefore, the entire emulation of G on H has constant slowdown. □

4.4. Emulating normal algorithms on the hypercube. Many practical computations on the hypercube are structured. The class of algorithms in which every node of the hypercube uses exactly one edge for communication at every time step and all of the edges used in a time step belong to the same dimension of the hypercube are called *leveled algorithms* (also known as *regular algorithms* [13]). A useful subclass of leveled algorithms are *normal algorithms*. A normal algorithm has the additional restriction that the dimensions used in consecutive time steps are consecutive. Many algorithms including bitonic sort, FFT, and tree-based algorithms like branch-and-bound can be implemented on the hypercube as normal algorithms [27]. An additional property of normal algorithms is that they can be emulated efficiently by bounded-degree networks such as the shuffle-exchange network and the butterfly. We state a result due to Schwabe [48] to this effect.

LEMMA 4.4.1. *An N-node butterfly can emulate any normal algorithm of an N-node hypercube with constant slowdown.*

We also require the following well-known result concerning the embedding of a butterfly in a hypercube. (See [20] for the stronger result that the butterfly is a subgraph of the hypercube.)

LEMMA 4.4.2. *An N-node butterfly can be embedded in an N-node hypercube with constant load, congestion, and dilation.*

THEOREM 4.4.3. *An N-node hypercube with $N^{1-\epsilon}$ worst-case faults (for any $\epsilon > 0$) can emulate T steps of any normal algorithm on an N-node fault-free hypercube in $O(T + \log N)$ steps.*

Proof. Let the faulty N-node hypercube be H and the fault-free N-node hypercube be G. H emulates any normal algorithm of G by using a sequence of constant slowdown emulations. Let an N-node butterfly B be embedded in H in the manner of Lemma 4.4.2. Any node of B that is mapped to a faulty node of H is considered faulty. (And H is not required to emulate these faulty nodes.) Since this is a constant load embedding, the number of faulty nodes in B is $O(N^{1-\epsilon})$. Clearly, H can emulate any computation of B with constant slowdown using the constant load, congestion, and dilation embedding of B in H. Let B' be a fault-free N-node butterfly. From Theorem 4.3.2, B can emulate B' with a constant slowdown. Now, from Lemma 4.4.1, B' can emulate any normal algorithm of G with a constant slowdown. Putting all these emulations together, we obtain a constant slowdown emulation of any normal algorithm on G on the faulty hypercube H. □

5. Random faults. In this section we show that an N-input host butterfly H can sustain many random faults and still emulate a fault-free N-input guest butterfly G with little slowdown. In particular, we show that if each node in H fails independently with probability $p = 1/\log^{(k)} N$, where $\log^{(k)}$ denotes the logarithm function

iterated k times, the slowdown of the emulation is $2^{O(k)}$, with high probability. For any constant k this slowdown is constant. Furthermore, for $k = O(\log^* N)$ the node failure probability p is constant, and the slowdown is $2^{O(\log^* N)}$. Previously, the most efficient self-emulation scheme known for an N-input butterfly required $\omega(\log \log N)$ slowdown [51].

The proof has the following outline. We begin by showing that the host H can emulate another N-input butterfly network B_k with constant slowdown. As in H, some of the nodes in B_k may fail at random (in which case it is not necessary for H to emulate them), but B_k is likely to contain fewer faults than H. In turn, B_k can emulate another butterfly B_{k-1} with even fewer faults. Continuing in this fashion, we arrive at B_1, which, with high probability, contains so few faults that it can emulate the guest G with constant slowdown. There are $k+1$ emulations, and each incurs a constant factor slowdown, so the total slowdown is $2^{O(k)}$.

5.1. Emulating a butterfly with fewer faults. We begin by explaining how H emulates B_k. The first step is to cover the N-input butterfly B_k with overlapping $(\log^{(k)} N)^2$-input subbutterflies. For ease of notation, let $M_k = (\log^{(k)} N)^2$. (For simplicity, we assume that $\log M_k$ is an integral multiple of 4.) For each i from 0 to $4(\log N/\log M_k) - 4$, there is a band of disjoint M_k-input subbutterflies spanning levels $(i \log M_k)/4$ through $((i+4)\log M_k)/4$. We call these subbutterflies the *band* i subbutterflies. Note that each band i subbutterfly shares $M_k^{3/4}$ rows with $M_k^{1/4}$ different band $i-1$ subbutterflies and $M_k^{3/4}$ rows with $M_k^{1/4}$ different band $i+1$ subbutterflies.

Each M_k-input subbutterfly in B_k is emulated by the corresponding subbutterfly in H. We say that an M_k-input subbutterfly in B_k *fails* if more than $\alpha\sqrt{M_k}\log M_k$ nodes inside the corresponding M_k-input subbutterfly in H fail, where α is a constant that will be determined later. If a subbutterfly in B_k fails, then H is not required to emulate any of the nodes that lie in that subbutterfly. As we shall see, if it does not fail, then the corresponding subbutterfly in H contains few enough faults that we can treat them as worst-case faults and apply the technique from section 4 to reconfigure around them. The following lemma bounds the probability that a subbutterfly in B_k fails.

LEMMA 5.1.1. *For $\alpha > 4e$, an M_k-input subbutterfly in B_k fails with probability at most $1/\log^{(k-1)} N$.*

Proof. An M_k-input subbutterfly fails if the corresponding M_k-input subbutterfly in H contains too many faults. An M_k-input subbutterfly in H contains a total of $M_k(1+\log M_k) \le 4(\log^{(k)} N)^2(\log^{(k+1)} N)$ nodes, each of which fails with probability $1/\log^{(k)} N = 1/\sqrt{M_k}$. Thus, the expected number of nodes that fail is at most $2\sqrt{M_k}\log M_k = 4(\log^{(k)} N)(\log^{(k+1)} N)$. Since each node fails independently, we can bound the probability that more than $\alpha\sqrt{M_k}\log M_k$ nodes fail using a Chernoff-type bound. For $\alpha > 4e$, the probability that more than $\alpha\sqrt{M_k}\log M_k$ nodes fail is at most $2^{-\alpha\sqrt{M_k}\log M_k}$ (for a proof, see [46]). Since $\alpha\sqrt{M_k}\log M_k > \log^{(k)} N$, this probability is at most $2^{-\log^{(k)} N} = 1/\log^{(k-1)} N$. □

The next lemma shows that if a subbutterfly in B_k does not fail, then the corresponding subbutterfly in H can emulate it with constant slowdown.

LEMMA 5.1.2. *If an M_k-input subbutterfly in B_k does not fail, then the corresponding subbutterfly in H can emulate it with constant slowdown.*

Proof. Since the number of faults in an M_k-input subbutterfly that does not fail is most $\alpha\sqrt{M_k}\log M_k$, we can treat them as worst-case faults and apply Theorem 4.3.2 with $\epsilon \approx 1/2$. □

The next lemma shows that the host H can emulate any computation performed by an N-input butterfly network B_k with constant slowdown. Recall that H is not required to emulate nodes in B_k that lie in subbutterflies in B_k that have failed.

LEMMA 5.1.3. *The host H can emulate B_k with constant slowdown.*

Proof. By Lemma 5.1.2, each M_k-input subbutterfly in B_k that has not failed can be emulated by the corresponding subbutterfly in H with constant slowdown using the technique of section 4. (Note that each node in B_k may be emulated by as many as four different subbutterflies in H.) In order to emulate the entire network B_k, it is also necessary to emulate the connections between the subbutterflies. As in section 4, let $M_k^{1-\epsilon}$ denote the number of faults in an M_k-input subbutterfly of H. For a subbutterfly that has not failed, $\epsilon \approx 1/2$. By Lemma 4.1.1, the number of rows in the subbutterfly containing either a fault or an input or output that is bad for Beneš routing is at most $M_k^{1-\epsilon}/2 + 4M_k^{1-2\epsilon/3}$, which is approximately $4M_k^{2/3}$. Each band i subbutterfly that does not fail shares $M_k^{3/4}$ rows with each of the band $i-1$ subbutterflies (and band $i+1$ subbutterflies) with which it overlaps. Thus, for each pair of overlapping butterflies that do not fail, most of the shared rows are both fault-free and good for routing in both subbutterflies. In order to emulate an M_k-input subbutterfly of H, the emulation strategy of section 4 covers it with smaller subbutterflies, each having $M_k^{\epsilon/2}$ inputs. If a smaller subbutterfly is used in the emulation, then none of the rows that pass through it contain either a fault or a bad input or output. Thus, the $M_k^{3/4}$ connections between each pair of overlapping M_k-input subbutterflies in bands i and $i-1$ can be emulated by routing constant congestion paths of length $O(\log M_k)$ through the shared rows. The rest of the proof is similar to that of Theorem 4.3.2. □

5.2. Emulating a series of butterflies. So far we have shown that the host H can emulate an N-input butterfly B_k that contains some faulty nodes. Although our ultimate goal is to show that H can emulate the guest network G, which contains no faulty nodes, we have made some progress. In the host network H, each node fails independently with probability $1/\log^{(k)} N$. In B_k, each $(\log^{(k)} N)^2$-input subbutterfly fails with probability $1/\log^{(k-1)} N$. A node in B_k fails if it lies in a subbutterfly that fails. Since each node in B_k lies in at most five $(\log^{(k)} N)^2$-input subbutterflies (once as an input, once as an output, and three times as an interior node), we have reduced the expected number of faults from $(N \log N)/\log^{(k)} N$ in H to fewer than $(5N \log N)/\log^{(k-1)} N$ in B_k.

The next step is to show that butterfly B_k can emulate a butterfly B_{k-1} with even fewer faults. In general, we cover butterfly B_j with $(\log^{(j)} N)^2$-input subbutterflies. For ease of notation, let $M_j = (\log^{(j)} N)^2$. We say that an M_j-input subbutterfly in B_j fails if the corresponding M_j-input subbutterfly in B_{j+1} contains more than $\alpha\sqrt{M_j}$ M_{j+1}-input subbutterflies that have failed. The following three lemmas are analogous to Lemmas 5.1.1 through 5.1.3.

LEMMA 5.2.1. *For $\alpha > 8e$, the probability that an M_j-input subbutterfly in B_j fails is at most $1/(\log^{(j-1)} N)$.*

Proof. The proof is by induction on j, starting with $j = k$ and working backward to $j = 0$. The base case is given by Lemma 5.1.1. For each value of i from 0 to $4(\log M_j/\log M_{j+1}) - 4$, there is a band of disjoint M_{j+1}-input subbutterflies in B_{j+1} that span levels $(i \log M_{j+1})/4$ through $((i+4)\log M_{j+1})/4$. These M_{j+1}-input subbutterflies can be partitioned into eight *classes* according to their band numbers.

Two bands of subbutterflies belong to the same class if their band numbers differ by a multiple of eight. There are at most

$$(M_j \log M_j)/(2M_{j+1} \log M_{j+1}) = (\log^{(j)} N)^2/2(\log^{(j+1)} N)(\log^{(j+2)} N)$$

subbutterflies in each of these classes, and within a class the subbutterflies are disjoint. By induction, each subbutterfly fails with probability at most $1/\log^{(j)} N$. Thus, in any particular class, the expected number of subbutterflies that fail is at most $(\log^{(j)} N)/2(\log^{(j+1)} N)(\log^{(j+2)} N)$, which is less than $\log^{(j)} N$. Using Chernoff-type bounds as in Lemma 5.1.1, for $\alpha > 16e$, the probability that more than $(\alpha \log^{(j)} N)/8 = (\alpha\sqrt{M_j})/8$ of these subbutterflies fail is at most $2^{-(\alpha \log^{(j)} N)/8}$, which is less than $1/8 \log^{(j-1)} N$. Thus, the probability that a total of $\alpha \log^{(j)} N$ subbutterflies fail in the eight classes is at most $1/\log^{(j-1)} N$. □

LEMMA 5.2.2. *If an M_j-input subbutterfly in B_j does not fail, then the corresponding M_j-input subbutterfly in B_{j+1} can emulate it with constant slowdown.*

Proof. If an M_j-input subbutterfly does not fail, then at most $\alpha\sqrt{M_j}$ of the overlapping M_{j+1}-input subbutterflies in the corresponding M_j-input subbutterfly in H fail. Each of these subbutterflies contains $M_{j+1}(1+\log M_{j+1}) \leq 4(\log^{(j+1)} N)^2 \log^{(j+2)} N$ nodes. Since the total number of nodes in all of these subbutterflies is at most $4\alpha \log^{(j)} N(\log^{(j+1)} N)^2 \log^{(j+2)} N$, i.e., approximately $\sqrt{M_j}$, we can treat them all as if they were worst-case faults and apply Theorem 4.3.2 with $\epsilon \approx 1/2$. □

LEMMA 5.2.3. *For $1 < j \leq k+1$, butterfly B_j can emulate B_{j-1} with constant slowdown.*

Proof. The proof is similar to the proof of Lemma 5.1.3. □

THEOREM 5.2.4. *For any fixed $\gamma > 0$ with probability at least $1 - 1/2^{N^{1-\gamma}}$, an N-input butterfly in which each node fails with probability $1/\log^{(k)} N$ can emulate a fault-free N-input butterfly with slowdown $2^{O(k)}$.*

Proof. The host network $H = B_{k+1}$ can emulate network B_1 with slowdown $2^{O(k)}$. In B_1, each subbutterfly with $(\log N)^2$ inputs fails with probability $1/\log^{(0)} N = 1/N$. Using Chernoff-type bounds as in Lemma 5.1.1, the probability that more than $N^{1-\gamma}$ of these subbutterflies fail is at most $1/2^{N^{1-\gamma}}$. If fewer than $N^{1-\gamma}$ of them fail, then we can treat the nodes contained in these subbutterflies as if they were worst-case faults. In this case, the total number of worst-case faults is at most $4N^{1-\gamma}(\log N)^2 \log\log N$. Hence, by applying Theorem 4.3.2 with $\epsilon \approx \gamma$, B_1 can emulate the guest network G with constant slowdown. □

6. Open problems. Some of the interesting problems left open by this paper are listed below.

1. Can a butterfly tolerate random faults with constant failure probability and still emulate a fault-free butterfly of the same size with constant slowdown?
2. Can an N-node butterfly (or any other N-node bounded-degree network) tolerate more than $N^{1-\epsilon}$ worst-case faults (e.g., $N/\log N$) and still emulate a fault-free network of the same type and size with constant slowdown?
3. Can a $2N$-input Beneš network tolerate $\Theta(N)$ worst-case switch failures and still route disjoint paths in any permutation between some set of $\Theta(N)$ inputs and outputs?
4. Can an N-input butterfly be embedded with constant load, congestion, and dilation in an N-input butterfly with more than $\log^{O(1)} N$ (e.g., $N^{1-\epsilon}$) worst-case faults?

Acknowledgments. Thanks to Bob Cypher and Joel Friedman for helpful discussions. We are also grateful to Bill Aiello for suggesting that our butterfly results imply results in fault-tolerant emulations of normal algorithms. The third author wishes to thank Bob Tarjan for his support and encouragement.

REFERENCES

[1] G. B. Adams, III and H. J. Siegel, *The extra stage cube: A fault-tolerant interconnection network for supersystems*, IEEE Trans. Comput., C–31 (1982), pp. 443–454.

[2] W. Aiello and T. Leighton, *Coding theory, hypercube embeddings, and fault tolerance*, in Proc. 3rd Annual ACM Symposium on Parallel Algorithms and Architectures, July 1991, pp. 125–136.

[3] W. A. Aiello, personal communication, July 1992.

[4] M. Ajtai, N. Alon, J. Bruck, R. Cypher, C. T. Ho, M. Naor, and E. Szemerédi, *Fault tolerant graphs, perfect hash functions and disjoint paths*, in Proc. 33rd Annual Symposium on Foundations of Computer Science, Oct. 1992, pp. 693–702.

[5] F. Annexstein, *Fault tolerance in hypercube-derivative networks*, Computer Architecture News, 19(1)(1991), pp. 25–34.

[6] S. Arora, T. Leighton, and B. Maggs, *On-line algorithms for path selection in a nonblocking network*, SIAM J. Comput., 25 (1996), pp. 600–625.

[7] S. Assaf and E. Upfal, *Fault-tolerant sorting network*, in Proc. 31st Annual Symposium on Foundations of Computer Science, Oct. 1990, pp. 275–284.

[8] Y. Aumann and M. Ben-Or, *Asymptotically optimal PRAM emulation on faulty hypercubes*, in Proc. 32nd Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Piscataway, NJ, Oct. 1991, pp. 440–457.

[9] S. N. Bhatt, F. R. K. Chung, F. T. Leighton, and A. L. Rosenberg, *Tolerating faults in synchronization networks*, Parallel Processing: CONPAR92-VAPPV, Lyon, France, Lecture Notes in Comput. Sci. 634, Springer, Berlin, 1992, pp. 1–12.

[10] J. Bruck, R. Cypher, and C.-T. Ho, *Fault-tolerant meshes with minimal numbers of spares*, IEEE Trans. Comput., 42 (1993), pp. 1089–1104.

[11] J. Bruck, R. Cypher, and C.-T. Ho, *Fault-tolerant meshes with small degree*, SIAM J. Comput., 26 (1997), pp. 1764–1784.

[12] J. Bruck, R. Cypher, and D. Soroker, *Running algorithms efficiently on faulty hypercubes*, Computer Architecture News, 19(1)(1991), pp. 89–96.

[13] J. Bruck, R. Cypher, and D. Soroker, *Tolerating faults in hypercubes using subcube partitioning*, IEEE Trans. Comput., 41 (1992), pp. 599–605.

[14] R. Cole, B. Maggs, and R. Sitaraman, *Reconfiguring arrays with faults part* I: *Worst-case faults*, SIAM J. Comput., 26 (1997), pp. 1581–1611.

[15] R. Cole, B. Maggs, and R. Sitaraman, *Routing on butterfly networks with random faults*, in Proc. 36th Annual Symposium on Foundations of Computer Science, Oct. 1995, pp. 558–570.

[16] S. Dutt and J. P. Hayes, *On designing and reconfiguring k-fault-tolerant tree architectures*, IEEE Trans. Comput., C–39 (1990), pp. 490–503.

[17] S. Dutt and J. P. Hayes, *Designing fault-tolerant systems using automorphisms*, J. Parallel and Distributed Computing, 12 (1991), pp. 249–268.

[18] S. Even and A. Litman, *Layered cross product—A technique to construct interconnection networks*, in Proc. 4th Annual ACM Symposium on Parallel Algorithms and Architectures, July 1992, pp. 60–69.

[19] M. R. Fellows, *Encoding Graphs in Graphs*, Ph.D. thesis, Department of Computer Science, University of California, San Diego, CA, 1985.

[20] D. S. Greenberg, L. S. Heath, and A. L. Rosenberg, *Optimal embeddings of butterfly-like graphs in the hypercube*, Math. Systems Theory, 23 (1990), pp. 61–77.

[21] R. I. Greenberg and C. E. Leiserson, *Randomized routing on fat-trees*, in Randomness and Computation, Advances in Computing Research, Vol. 5, S. Micali, ed., JAI Press, Greenwich, CT, 1989, pp. 345–374.

[22] J. W. Greene and A. El Gamal, *Configuration of VLSI arrays in the presence of defects*, J. Assoc. Comput. Mach., 31 (1984), pp. 694–717.

[23] J. Hastad, T. Leighton, and M. Newman, *Reconfiguring a hypercube in the presence of faults*, in Proc. 19th Annual ACM Symposium on Theory of Computing, May 1987, pp. 274–284.

[24] J. Hastad, T. Leighton, and M. Newman, *Fast computation using faulty hypercubes*, in Proc. 21st Annual ACM Symposium on Theory of Computing, May 1989, pp. 251–263.

[25] C. Kaklamanis, A. R. Karlin, F. T. Leighton, V. Milenkovic, P. Raghavan, S. Rao, C. Thomborson, and A. Tsantilas, *Asymptotically tight bounds for computing with faulty arrays of processors*, in Proc. 31st Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Piscataway, NJ, Oct. 1990, pp. 285–296.

[26] R. Koch, T. Leighton, B. Maggs, S. Rao, and A. Rosenberg, *Work-preserving emulations of fixed-connection networks*, J. Assoc. Comput. Mach., 44 (1997), pp. 104–147.

[27] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes*, Morgan Kaufmann, San Mateo, CA, 1992.

[28] F. T. Leighton and B. M. Maggs, *Fast algorithms for routing around faults in multibutterflies and randomly-wired splitter networks*, IEEE Trans. Comput., 41 (1992), pp. 578–587.

[29] F. T. Leighton, B. M. Maggs, A. G. Ranade, and S. B. Rao, *Randomized routing and sorting on fixed-connection networks*, J. Algorithms, 17 (1994), pp. 157–205.

[30] F. T. Leighton, B. M. Maggs, and S. B. Rao, *Packet routing and job-shop scheduling in O(congestion + dilation) steps*, Combinatorica, 14 (1994), pp. 167–180.

[31] T. Leighton and C. E. Leiserson, *Wafer-scale integration of systolic arrays*, IEEE Trans. Comput., C–34 (1985), pp. 448–461.

[32] T. Leighton, Y. Ma, and C. G. Plaxton, *Breaking the $\Theta(n \log^2 n)$ barrier for sorting with faults*, J. Comput. System Sci., 54 (1997), pp. 265–304.

[33] C. E. Leiserson, *Fat-trees: Universal networks for hardware-efficient supercomputing*, IEEE Trans. Comput., C–34 (1985), pp. 892–901.

[34] G. Lin, *Fault tolerant planar communication networks*, in Proc. 24th Annual ACM Symposium on the Theory of Computing, May 1992, pp. 133–139.

[35] M. Livingston, Q. Stout, N. Graham, and F. Hararay, *Subcube Fault-Tolerance in Hypercubes*, Tech. report CRL-TR-12-87, University of Michigan Computing Research Laboratory, Ann Arbor, Sept. 1987.

[36] Y.-D. Lyuu, *Fast fault-tolerant parallel communication and on-line maintenance using information dispersal*, Math. Systems Theory, 24 (1991), pp. 273–294.

[37] B. M. Maggs and R. K. Sitaraman, *Simple algorithms for routing on butterfly networks with bounded queues*, in Proc. 24th Annual ACM Symposium on the Theory of Computing, May 1992, pp. 150–161; SIAM J. Comput., to appear.

[38] F. Meyer auf der Heide, *Efficiency of universal parallel computers*, Acta Inform., 19 (1983), pp. 269–296.

[39] F. Meyer auf der Heide, *Efficient simulations among several models of parallel computers*, SIAM J. Comput., 15 (1986), pp. 106–119.

[40] F. Meyer auf der Heide and R. Wanka, *Time-optimal simulations of networks by universal parallel computers*, in Proc. 6th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 349, Springer, Berlin, 1989, pp. 120–131.

[41] D. C. Opferman and N. T. Tsao-Wu, *On a class of rearrangeable switching networks–part* II*: Enumeration studies and fault diagnosis*, Bell System Tech. J., 50 (1971), pp. 1601–1618.

[42] N. Pippenger, *Telephone switching networks*, in Proc. Symposia in Applied Mathematics, Vol. 26, American Mathematical Society, Providence, RI, 1982, pp. 101–133.

[43] N. Pippenger and G. Lin, *Fault-tolerant circuit-switching networks*, in Proc. 4th Annual ACM Symposium on Parallel Algorithms and Architectures, June 1992, pp. 229–235.

[44] M. O. Rabin, *Efficient dispersal of information for security, load balancing, and fault tolerance*, J. Assoc. Comput. Mach., 36 (1989), pp. 335–348.

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

[46] P. Raghavan, *Lecture Notes on Randomized Algorithms*, Research Report RC 15340 (#68237), IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY, Jan. 1990.

[47] E. J. Schwabe, *On the computational equivalence of hypercube-derived networks*, in Proc. 2nd Annual ACM Symposium on Parallel Algorithms and Architectures, July 1990, pp. 388–397.

[48] E. J. Schwabe, *Efficient Embeddings and Simulations for Hypercubic Networks*, Ph.D. thesis, Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA, June 1991.

[49] S. Sowrirajan and S. M. Reddy, *A design for fault-tolerant full connection networks*, in Proc. International Conference on Science and Systems, IEEE Computer Society Press, Piscataway, NJ, Mar. 1980, pp. 536–540.

[50] J. Spencer, *Ten Lectures on the Probabilistic Method*, SIAM, Philadelphia, PA, 1987.

[51] H. Tamaki, *Efficient self-embedding of butterfly networks with random faults*, in Proc. 33rd Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Piscataway, NJ, Oct. 1992, pp. 533–541.

[52] H. Tamaki, *Robust bounded-degree networks with small diameters*, in Proc. 4th Annual ACM Symposium on Parallel Algorithms and Architectures, June 1992, pp. 247–256.
[53] S. Toledo, *Competitive fault-tolerance in area-universal networks*, in Proc. 4th Annual ACM Symposium on Parallel Algorithms and Architectures, June 1992, pp. 236–246.
[54] L. G. Valiant, *A scheme for fast parallel communication*, SIAM J. Comput., 11 (1982), pp. 350–361.
[55] A. Waksman, *A permutation network*, J. Assoc. Comput. Mach., 15 (1968), pp. 159–163.
[56] A. Wang and R. Cypher, *Fault-Tolerant Embeddings of Rings, Meshes and Tori in Hypercubes*, Tech. report IBM RJ 8569, IBM Almaden Research Center, San Jose, CA, Jan. 1992.
[57] A. Wang, R. Cypher, and E. Mayr, *Embedding complete binary trees in faulty hypercubes*, in Proc. 3rd IEEE Symposium on Parallel and Distributed Processing, IEEE Computer Society Press, Piscataway, NJ, Dec. 1991, pp. 112–119.

SIAM J. COMPUT.
Vol. 27, No. 5, pp. 1334–1347, October 1998
© 1998 Society for Industrial and Applied Mathematics
008

ON THE COMPLEXITY OF NEGATION-LIMITED BOOLEAN NETWORKS*

ROBERT BEALS†, TETSURO NISHINO‡, AND KEISUKE TANAKA§

Abstract. A theorem of Markov precisely determines the number r of NEGATION gates necessary and sufficient to compute a system of boolean functions F. For a system of boolean functions on n variables, $r \leq b(n) = \lceil \log_2(n+1) \rceil$. We call a circuit using $b(n)$ NEGATION gates *negation-limited*. We continue recent investigations into negation-limited circuit complexity, giving both upper and lower bounds.

A circuit with inputs $x_1, \ldots, x_n$ and outputs $\neg x_1, \ldots, \neg x_n$ is called an *inverter*, for which $r = \lceil \log_2(n+1) \rceil$. Fischer has constructed negation-limited inverters of size $O(n^2 \log n)$ and depth $O(\log n)$. Recently, Tanaka and Nishino have reduced the circuit size to $O(n \log^2 n)$ at the expense of increasing the depth to $\log^2 n$. We construct negation-limited inverters of size $O(n \log n)$, with depth only $O(\log n)$, and we conjecture that this is optimal. We also improve a technique of Valiant for constructing monotone circuits for slice functions (introduced by Berkowitz).

Next, we introduce some lower bound techniques for negation-limited circuits. We provide a $5n + 3\log(n+1) - c$ lower bound for the size of a negation-limited inverter. In addition, we show that for two different restricted classes of circuit, negation-limited inverters require superlinear size.

Key words. circuit complexity, negation-limited circuit, upper bounds, lower bounds

AMS subject classifications. 68Q10, 68Q15

PII. S0097539794275136

1. Introduction.

1.1. Background. The theory of monotone boolean circuit complexity has met with considerable success [15, 3, 19, 7, 14]. Good lower bounds for both size and depth of monotone circuits (i.e., circuits without NEGATION gates) computing many explicit functions are now known. However, the theory of general boolean circuits, *with* NEGATION gates, is much less well understood. Exponential gaps between monotone and general circuit complexity are known, for both circuit size [19] and circuit depth [14]. The effect of NEGATION gates on circuit complexity remains to a large extent a mystery.

This motivates the study of *negation-limited* circuit complexity: what is the effect on circuit complexity of restricting the number of negations available? Markov [11] gives an explicit formula (see below) for the maximum number r of NEGATION gates *required* to compute a system of boolean functions F, without regard to circuit complexity. The maximum value of r for a circuit with n inputs is $\lceil \log(n+1) \rceil$, the number of bits in the binary representation of n (all logarithms in this paper are base two).

*Received by the editors October 3, 1994; accepted for publication (in revised form) July 23, 1996; published electronically May 19, 1998. These results have been previously announced in Proc. 26th Annual ACM Symp. on Theory of Computing, 1994, pp. 38–47 and Proc. 27th Annual ACM Symp. on Theory of Computing, 1995, pp. 585–595.

http://www.siam.org/journals/sicomp/27-5/27513.html

† School of Mathematics, Institute for Advanced Study, Princeton, NJ 08540 and DIMACS, Rutgers University, P.O. Box 1179, Piscataway, NJ 08855 (beals@math.ias.edu). This research was supported by an NSF Mathematical Sciences Postdoctoral Fellowship and by the Alfred P. Sloan Foundation.

‡ Department of Communications and Systems Engineering, The University of Electro-Communications, Chofu-shi, Tokyo 182, Japan (nishino@sw.cas.uec.ac.jp).

§ NTT Information and Communication Systems Laboratories, 1-1 Hikarinooka Yokosuka-shi, Kanagawa 239, Japan (keisuke@sucaba.isl.ntt.co.jp). This work was done in part at Japan Advanced Institute of Science and Technology, Tatsunokuchi, Ishikawa, Japan.

We shall denote this number by $b(n)$. Fischer [8] shows that restricting the number of negations in a circuit to $b(n)$ entails only a polynomial blowup in circuit size. This is in sharp contrast to the situation for monotone circuits. É. Tardos [19] has shown that there is an exponential gap between the general complexity and the monotone complexity of some monotone functions. In related work, Santha and Wilson [16] have studied the negation-limited complexity of constant depth circuits, obtaining both upper and lower bounds for the number of NEGATION gates required by circuits of a given depth.

In a preliminary version of this paper, Tanaka and Nishino [18] have found an alternative to Fischer's construction; they decrease the size of the circuits at the expense of increasing the depth. We further reduce the size, while simultaneously reducing the depth to that of Fischer. We also exhibit a relationship between the negation-limited complexity of the inverter (see below) and the monotone complexity of certain types of monotone functions.

1.2. Definitions and preliminaries. Let F be a collection of boolean functions $f_1, \ldots, f_m$ defined on $\{0,1\}^n$. We denote by $C(F)$ or $C(f_1, \ldots, f_m)$ the *circuit complexity* of F, i.e., the size (number of gates) of the smallest circuit of AND, OR, and NEGATION gates with inputs

$$x_1, \ldots, x_n$$

and outputs

$$f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n).$$

We call a circuit with no more than r NEGATION gates an r-circuit. We denote by $C^r(F)$ or $C^r(f_1, \ldots, f_m)$ the size of the smallest r-circuit computing F. If the system of functions cannot be computed with only r NEGATION gates, then $C^r(F)$ is undefined. Similarly, we denote by $D^r(F)$ the minimum *depth* of an r-circuit computing F. The *inverter* is the collection I_n of functions $f_1, \ldots, f_n$, where for all $1 \le i \le n$, $f_i(x_1, \ldots, x_n) = \neg x_i$. For $0 \le k \le n$, the (k,n)-*inverter* is the collection I_n^k of functions $f_1, \ldots, f_n$, where for all $1 \le i \le n$,

$$f_i(x_1, \ldots, x_n) = \begin{cases} 0 & \text{if } \sum_{j=1}^n x_j < k, \\ \neg x_i & \text{if } \sum_{j=1}^n x_j = k, \\ 1 & \text{if } \sum_{j=1}^n x_j > k. \end{cases}$$

Let F be a system of n-input boolean functions. A *chain* C in the boolean lattice $\{0,1\}^n$ is an increasing sequence $a^1 < \cdots < a^k \in \{0,1\}^n$. The *decrease* of F on C is the number of $i \le k$ such that for some j, $f_j(a^{i-1}) > f_j(a^i)$. We define $d(F)$ to be the maximum decrease of F on any chain C. Note that $d(F) \le n$, and this is attained for $F = I_n$. Markov [11] has shown that $b(d(F))$ NEGATION gates are necessary and sufficient to compute any system $f_1, \ldots, f_m$ of functions. Thus, $C^r(F)$ is always defined for $r \ge b(n)$. We call $C^{b(d(F))}(F)$ the *negation-limited* complexity of the system of functions F.

1.3. Main results. We give improved upper bounds and lower bounds on the negation-limited complexity of the inverter.

THEOREM 1.1. *The negation-limited complexity of the inverter I_n is $O(n \log n)$. In fact, I_n may be computed by negation-limited circuits of size $O(n \log n)$ and depth $O(\log n)$.*

This in turn yields upper bounds on $C^{b(n)}(F)$ for an arbitrary system of functions F, via the standard technique of using DeMorgan's laws to push all negations in a circuit to the inputs.

COROLLARY 1.2. *For any system F of n-input boolean functions,*

$$C^{b(n)}(F) \leq 2C(F) + O(n \log n). \qquad \square$$

Currently no superlinear lower bound for the general circuit complexity of an explicit Boolean function is known. The above theorem and corollary imply that if we can show an $\omega(n \log n)$ lower bound on $C^{b(n)}(f)$ for some explicit Boolean function f, we also obtain an $\omega(n \log n)$ lower bound on the general circuit complexity of f.

Markov [11] constructs inverters using monotone functions and $b(n)$ negations, but he does not consider the complexity of the monotone functions that he uses. Akers [2] (see [12]) gives the first explicit construction of a negation-limited inverter. His circuit uses $b(n)$ NEGATION gates and positive weight threshold gates, and has size $O(n)$ and depth $O(\log n)$. For the remainder of this paper, we restrict our attention to circuits of AND, OR, and NEGATION gates. Fischer [8] gives such a circuit for I_n with size $O(n^2 \log^2 n)$ and depth $O(\log^2 n)$, using only $b(n)$ NEGATION gates. Sorting networks play a key role in Fischer's construction (and in all subsequent constructions). Using the sorting network of Ajtai, Komlós, and Szemerédi [1] (see [13]) in Fischer's construction reduces the size to $O(n^2 \log n)$ and the depth to $O(\log n)$. In a preliminary version of this paper, Tanaka and Nishino [18] have investigated the negation-limited complexity of the inverter, giving an upper bound of $O(n \log^2 n)$, using a construction of depth $\theta(\log^2 n)$.

Our circuit, with size $O(n \log n)$ and depth $O(\log n)$, improves on previous negation-limited circuits for I_n by a factor of at least $(\log n)$ in size. Note that the only previous construction with logarithmic depth had superquadratic size.

A circuit with no NEGATION gates is *monotone*. A *monotone function* f is a boolean function which satisfies $f(x_1, \ldots, x_n) \leq f(y_1, \ldots, y_n)$ whenever $x_i \leq y_i$ for all i. Monotone functions are exactly those computed by monotone circuits. Observe that the (k, n)-inverter is a system of monotone functions. Berkowitz [6] shows that the monotone complexity of the (k, n)-inverter is polynomial, with a construction of size $O(n^2 \log n)$ and depth $O(\log n)$. Valiant [20] constructs monotone (k, n)-inverters of size $O(n \log^2 n)$ and depth $O(\log^2 n)$.

THEOREM 1.3. *The monotone complexity of the (k, n)-inverter is $O(n \log n)$. In fact, I_n^k has monotone circuits of size $O(n \log n)$ and depth $O(\log n)$.*

We also show (see Theorem 4.4) that there is a close relationship between $C^0(I_n^k)$ and $C^{b(n)}(I_n)$. (For more background on monotone circuits, we refer to section 4 of Boppana and Sipser's article [7].)

We also obtain several lower bounds. For arbitrary negation-limited inverters, we obtain the following theorem.

THEOREM 1.4. *Let n be one less than a power of 2. Then any negation-limited circuit for I_n has size $\geq 5n + 3\log(n+1) - c$ and depth $\geq 4\log(n+1) - c$.*

In addition, we show that for two different restricted classes of circuits, negation-limited inverters require size $\Omega(n \log n)$.

The remainder of this paper is organized as follows. In section 2, we describe elements of previous constructions of negation-limited inverters which are common to ours. In section 3, we give the heart of our construction of negation-limited inverters.

In section 4, we show the relationship between negation-limited inverters and monotone (k, n)-inverters. Section 5 gives the main technical lemma on functions computed at the NEGATION gates of negation-limited inverters, and proves Theorem 1.4. In section 6, we prove superlinear bounds for various restricted classes of negation-limited inverters. We conclude in section 7 by mentioning some open problems.

2. The Fischer and Tanaka–Nishino inverters. We review briefly the circuits of Fischer [8] and Tanaka and Nishino [18]. These circuits have several elements in common with each other, and with ours as well. The most important of these is a sorting network.

A *comparator* is a circuit element with two inputs and two outputs. The inputs are taken from some ordered set U. The first output of the comparator is the maximum of the two inputs, while the second output is the minimum of the two inputs. A sorting network is a circuit, with n inputs and n outputs, composed entirely of comparators. For any n-tuple $x_1, \ldots, x_n$ of inputs from U, the n outputs $y_1, \ldots, y_n$ of the sorting network are a permutation of the x_i and are in decreasing order. Ajtai, Komlós, and Szemerédi [1] (see [13]) have constructed sorting networks using $O(n \log n)$ comparators, organized in $O(\log n)$ levels of $\lfloor n/2 \rfloor$ comparators each. Note that if the set U is $\{0, 1\}$, then a comparator can be constructed without negations. Indeed, for $x, y \in \{0, 1\}$, $\min(x, y) = x \wedge y$ and $\max(x, y) = x \vee y$. Thus, the Ajtai–Komlós–Szemerédi result yields a monotone circuit of size $O(n \log n)$ and depth $O(\log n)$ for sorting n bits.

Both Fischer and Tanaka and Nishino first sort the n input bits $x_1, \ldots, x_n$. The outputs $y_1, \ldots, y_n$ of the sorting network are fed into a subcircuit M_n (due to Fischer [8]) with the following properties.

1. M_n has n binary inputs $y_1, \ldots, y_n$ and n outputs $z_1, \ldots, z_n$.
2. M_n has size $O(n)$, depth $O(\log n)$, and uses $b(n)$ NEGATION gates.
3. If $y_1 \geq y_2 \geq \cdots \geq y_n$ then $z_i = \neg y_i$ for $1 \leq i \leq n$.

The negations of the x_i may now be calculated monotonically from the x_i, the y_i, and the z_i. Fischer does this using $O(n)$ sorting networks in parallel, resulting in a circuit of size $O(n^2 \log n)$ and depth $O(\log n)$ if the Ajtai–Komlós–Szemerédi sorting network is used. Tanaka and Nishino, on the other hand, use the monotone $(n, 2n)$-inverter of Valiant [20] which inverts $2n$ inputs provided that exactly n of them equal 1 (as is the case with $x_1, \ldots, x_n, z_1, \ldots, z_n$). Valiant's circuit is monotone, with size $O(n \log^2 n)$ and depth $O(\log^2 n)$.

Note that in both the Fischer and the Tanaka–Nishino constructions, the circuit is arranged in three phases: the sorting network, the subcircuit M_n, and the top phase which computes the final answer. (We view circuits as having their inputs at the bottom and outputs at the top, so the final phase of a circuit with several phases is the top phase.) The bottom two phases require only size $O(n \log n)$ and depth $O(\log n)$ (in fact, the second phase has linear size). Also, both the Fischer and the Tanaka–Nishino constructions require no more than the x_i, the y_i, and the z_i as inputs to the top phase. We show that by allowing the top phase access to intermediate results from the sorting network (first phase), the top phase may be computed by a size $O(n \log n)$, depth $O(\log n)$ circuit. Our top phase may be thought of as an "upside-down" sorting network. We describe this in the next section.

3. Description of the inverter. Our negation-limited circuit for I_n, like its predecessors, begins with a sorting network and Fischer's M_n circuit. We present a new approach to the third phase, which achieves considerable savings in complexity

over previous constructions by means of a novel application of the Ajtai–Komlós–Szemerédi result.

Before going into the details of the circuit, we briefly describe the intuition. The differences between the three constructions ([8, 18] and ours) seem to stem from how one thinks of the outputs of the M_n subcircuit. Of course, in all three constructions, the outputs of M_n are computing exactly the same system of n functions. Nevertheless, these outputs are treated completely differently by the three approaches. In Fischer's original negation-limited inverter [8], the inputs and outputs of M_n are thought of as the threshold functions (of $x_1, \ldots, x_n$) and their negations, respectively. Fischer uses these bits to select the output of an appropriate subcircuit, according to the number k of ones in the input. There are $n+1$ such subcircuits, one for each $0 \le k \le n$. Tanaka and Nishino [18] avoid having to consider $(n+1)$ different cases in parallel by noticing that the x_i, together with the outputs of M_n, are a string of $2n$ bits containing exactly n ones (such a string can be inverted monotonically using Valiant's monotone $(n, 2n)$-inverter [20]). In their construction, the outputs of M_n are thought of as *balancing* the inputs $x_1, \ldots, x_n$. We take a different view. We view the outputs of M_n as being a *permutation* of the negations of the x_i. It is the task of the third phase to *rearrange* these bits into their proper position. This rearrangement essentially undoes the sorting performed by the first phase, one level at a time.

We now present the details. Let ℓ be the depth of the sorting network used in the first phase. For each level in the sorting network, the third phase has a corresponding level, composed of a monotone, linear size, constant depth circuit. The ordering of the levels, however, is reversed between the first phase and the third phase. That is, we number the levels in the sorting network from the bottom (numbered 1) to the top (numbered ℓ), and we number the levels in the third phase from 1 at the top to ℓ at the bottom. When numbered in this way, level i of the sorting network corresponds to level i of the third phase. We will use S_i to denote level i of the sorting network, and T_i to denote level i of the top phase.

For $1 \le i \le \ell$, let $x_1^i, \ldots, x_n^i$ denote the inputs to S_i, and let $x_1^{\ell+1}, \ldots, x_n^{\ell+1}$ denote the outputs of S_ℓ. The subcircuit T_i receives $2n$ input bits: the n input bits $x_1^i, \ldots, x_n^i$ of S_i, and the negations $\neg x_1^{i+1}, \ldots, \neg x_n^{i+1}$ of the n output bits of S_i. For $1 \le j \le n$, $\neg x_j^{i+1}$ is computed one level below by T_{i+1}, or by M_n in the case $i = \ell$. Using only linear size, constant depth, monotone circuitry, T_i outputs the negations $\neg x_1^i, \ldots, \neg x_n^i$ of the n input bits to S_i. If $i > 1$, these are passed up to the next level, T_{i-1}. Note that at the top level, T_1 outputs the negations of the inputs to S_1, i.e., T_1 outputs $\neg x_1, \ldots, \neg x_n$. The overall structure of the circuit is shown in Figure 1. Note that, for readability, the wires connecting the outputs of S_{i-1} to the inputs of T_i are not shown.

To complete the description of the circuit for I_n, we need to describe a monotone, linear size, constant depth circuit for T_i. Since S_i is simply $\lfloor n/2 \rfloor$ comparators operating in parallel, it suffices to give a (constant size and depth) monotone circuit for each comparator. This we do in the following lemma.

LEMMA 3.1. *Let x and y be boolean variables, and let $u = \neg \max(x, y)$ and $v = \neg \min(x, y)$. Then $\neg x$ and $\neg y$ may be computed monotonically from x, y, u, and v.*

Proof. A quick check shows that $\neg x = u \vee (y \wedge v)$ and $\neg y = u \vee (x \wedge v)$. □

Suppose that S_i has a comparator with inputs $x_{j_1}^i, x_{j_2}^i$ and outputs $x_{k_1}^{i+1}, x_{k_2}^{i+1}$. Then T_i will have a corresponding circuit element with inputs

$$\neg x_{k_1}^{i+1}, \neg x_{k_2}^{i+1}, x_{j_1}^i, x_{j_2}^i$$

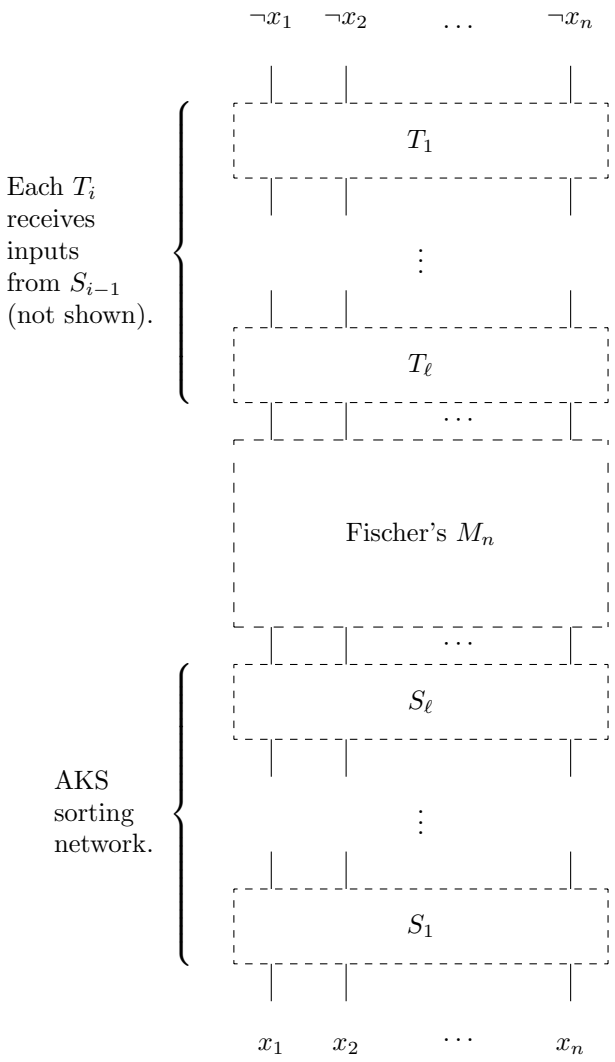


FIG. 1. *Overall structure of the circuit.*

and outputs

$$\neg x^i_{j_1}, \neg x^i_{j_2}.$$

By the above lemma, this can be done monotonically. This is illustrated in Figure 2. As S_i is a linear size, depth 1 circuit of comparators, we can construct T_i to be a monotone, linear size, depth 2 circuit.

We are now ready to prove Theorem 1.1.

Proof. We describe the circuit. As in Fischer [8] and Tanaka and Nishino [18], the input variables $x_1, \ldots, x_n$ are the inputs of a sorting network with ℓ levels of $O(n)$ gates each. By Ajtai, Komlós, and Szemerédi [1], we may assume $\ell = O(\log n)$. Still following Fischer and Tanaka and Nishino, the outputs $y_1, \ldots, y_n$ of the sorting network are the inputs to Fischer's M_n circuit, which has size $O(n)$ and depth $O(\log n)$

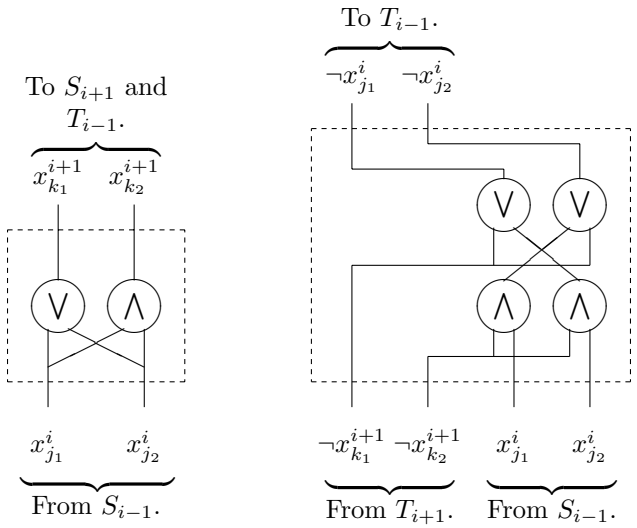


FIG. 2. *Corresponding circuit elements of S_i (left) and T_i (right).*

and uses $b(n)$ NEGATION gates. Since the inputs to M_n are sorted, the outputs $z_1, \ldots, z_n$ satisfy $z_i = \neg y_i$ for $i = 1, \ldots, n$.

From the z_i, and the inputs to the various levels of the sorting network, the subcircuits $T_\ell, \ldots, T_1$ described above compute the negations of the inputs to successively lower levels in the sorting network. It is clear by induction on $(\ell - i)$ that T_i outputs the negations $\neg x^i_1, \ldots, \neg x^i_n$ of the inputs to S_i. At the top of our circuit, T_1 computes the negations $\neg x_1, \ldots, \neg x_n$ of the input variables.

The circuit has depth $3\ell + O(\log n) = O(\log n)$. As each level of the circuit has linear size, the total number of gates used is $O(n \log n)$. The only NEGATION gates used are the $b(n)$ NEGATION gates used by M_n, as desired. □

4. Monotone (k, n)-inverters. Berkowitz [6] introduces the notion of a *slice function*. The boolean function $f : \{0,1\}^n \to \{0,1\}$ is a kth slice if

$$f(x_1, \ldots, x_n) = \begin{cases} 0 & \text{if } \sum_{i=1}^n x_i < k, \\ 1 & \text{if } \sum_{i=1}^n x_i > k. \end{cases}$$

For input strings containing exactly k ones, the behavior of f is not constrained. Any slice function is monotone. Note that an alternative, nonconstructive proof of Lemma 3.1 may be obtained by observing that, of the four bits x, y, u, v, exactly two are equal to one, so any function of x, y, u, v may be extended to a (monotone) slice function. The importance of monotone (k, n)-inverters is underscored by the following.

PROPOSITION 4.1. *For any system F of n-input kth slice functions,*

$$C^0(F) \leq 2C(F) + C^0(I^k_n). \qquad \square$$

The following allows us to focus on the behavior of our circuit on inputs with exactly k ones.

PROPOSITION 4.2. *I^k_n is the unique system of monotone functions which agrees with I_n on all inputs with exactly k ones.*

Proof. Suppose that $f_1, \ldots, f_n$ are a system of monotone functions which agree with I_n on all inputs with exactly k ones. Suppose that the number of ones in $x_1, \ldots, x_n$ is strictly greater than k. Let i be any integer between 1 and n. We must show that $f_i(x_1, \ldots, x_n) = 1$. By the monotonicity of f_i, it suffices to exhibit $y_1, \ldots, y_n$ such that $f_i(y_1, \ldots, y_n) = 1$ and for all $1 \leq j \leq n$, $y_j \leq x_j$. For this, we assign the y_j in such a way that exactly k of them are one, $y_i = 0$, and each $y_j \leq x_j$ (we obtain the y_j by changing some of the x_j from one to zero, including x_i if it is nonzero, until exactly k bits are one). The proof for the case that fewer than k of the x_j are one is analogous. □

Berkowitz [6] constructs monotone (k, n)-inverters of size $O(n^2 \log n)$ and depth $O(\log n)$. Valiant [20] gives monotone circuits of size $O(n \log^2 n)$ and depth $O(\log^2 n)$ for the (k, n)-inverter (see also Wegener [21, 22]). Our techniques can be used to improve on Valiant's results by a factor of $(\log n)$ in size and depth. We present the proof of Theorem 1.3.

Proof. We use our negation-limited circuit for I_n, replacing the M_n subcircuit by a circuit which outputs k zeroes followed by $(n - k)$ ones. Note that the only negations in our circuit for I_n occur in the M_n subcircuit. Also, if there are exactly k ones among the inputs to our negation-limited inverter, then the M_n subcircuit will output k zeroes followed by $(n - k)$ ones. Therefore, the new circuit will agree with I_n (i.e., it will compute the negations of its input bits) whenever there are exactly k ones in the input. By Proposition 4.2, we are done. □

Since the inputs to M_n are sorted, the functions computed by the NEGATION gates depend only on the number of ones in the input. Therefore, in the above construction, instead of replacing the outputs of M_n by constants, we could have replaced the outputs of the NEGATION gates by constants. One might ask whether it is always possible to obtain a monotone (k, n)-inverter from a negation-limited inverter by replacing the negation gates with constants. We shall see (perhaps surprisingly) that something close to this is true. Before formalizing this, we quote the following lemma of Tanaka and Nishino (see [18, Lemma 4.2]).

LEMMA 4.3 (see Tanaka and Nishino). *Suppose $n + 1$ is a power of 2, and let f be the function computed by some NEGATION gate of a negation-limited circuit for I_n. Then $f(x_1, \ldots, x_n)$ depends only on the number of i such that $x_i = 1$.* □

We will prove a stronger version of this lemma later (see Lemma 5.1).

Now we can make precise the relationship between negation-limited inverters and monotone (k, n)-inverters.

THEOREM 4.4. *Let n be a positive integer. Then $C^0(I_n^k) \leq C^{b(n')}(I_{n'})$, for some $n' < 2n$. In fact, a monotone (k, n)-inverter can be constructed with the same size and depth as a negation-limited circuit for $I_{n'}$.*

Proof. Let $n' = 2^r - 1$, where 2^r is the smallest power of 2 greater than n. Note that 2^r is at most $2n$. Let Δ be any negation-limited circuit for $I_{n'}$. By Lemma 4.3, if the number of ones in the input is constant, then the NEGATION gates in Δ compute constant functions. Replacing these gates by appropriate (according to k) constants yields a monotone (k, n')-inverter. To obtain a monotone (k, n)-inverter, we simply fix the first $(n' - n)$ input bits to 0 and ignore the first $(n' - n)$ output bits. This yields a monotone circuit Δ' computing I_n^k with the same size and depth as Δ. □

In fact, *all* known constructions of monotone (k, n)-inverters may be seen as negation-limited inverters with the NEGATION gates replaced by constants. This modification applied to Fischer's circuit for I_n (using the Ajtai, Komlós, and Szemerédi sorting network [1]) yields Berkowitz's (k, n)-inverter. There is no previously

published negation-limited inverter corresponding to Valiant's (k, n)-inverter, so we describe one here. First we give a sketch of Valiant's construction.

Valiant's circuit is very similar to ours (with the M_n removed), except that instead of using comparators, *merging networks* are used. A merging network takes as inputs two sorted lists u and v, and outputs the list $w = \text{merge}(u, v)$ obtained by sorting the concatenated list uv. Batcher [4] shows that two lists of m elements can be merged by a network of $O(m \log m)$ comparators. Thus, if u and v are binary strings of length m, they can be merged by a monotone circuit of size $O(m \log m)$ (this circuit has depth $O(\log m)$). If u is a binary string, we denote by $\neg u$ the string obtained from u by replacing each symbol by its negation, and we denote by $\text{reverse}(u)$ the string obtained from u by reversing the order of the symbols. Note that if u is in sorted (i.e., decreasing) order, then so is $\text{reverse}(\neg u)$.

The bottom half of Valiant's circuit is a sorting network, built out of $\log n$ layers of merging networks (the ith layer merges pairs of consecutive substrings of length 2^i). The top half is analogous to our top phase, with the following taking the place of Lemma 3.1.

LEMMA 4.5 (Valiant). *Suppose that the sorted binary strings u of length s and v of length t, together with the string $w = \text{reverse}(\neg \text{merge}(u, v))$, are given. Then* $\text{reverse}(\neg u)$ *and* $\text{reverse}(\neg v)$ *may be calculated by merging networks.*

Proof. By symmetry, we only need to consider $\text{reverse}(\neg u)$. This is easily seen to be the middle s bits of $\text{merge}(v, w)$. □

(Note that the case $s = t = 1$ of Valiant's lemma provides yet another proof of Lemma 3.1.)

Since k is fixed, the negations of the output bits of the top layer of the bottom half are constants. (Actually, this top layer of the bottom half is superfluous for the monotone (k, n) inversion problem, and Valiant does without it. We leave it in for simplicity, and because it is necessary if we are to obtain a negation-limited inverter.) Successively higher layers of the top half calculate the reverses of the negations of inputs to merging networks in successively lower layers of the bottom half. At the top layer of the top half, the negations of the input bits are calculated (reversing a string of length one has no effect).

To obtain a negation-limited inverter from Valiant's circuit, we simply feed the output of the bottom half to Fischer's M_n circuit, which computes the negation of the output of the top layer of the bottom half. This is then given as input to the top half. (The circuit constructed in this way is about half the size of that of Tanaka and Nishino [18], since we modify Valiant's (k, n)-inverter instead of using Valiant's $(n, 2n)$-inverter.) Therefore, Valiant's techniques for constructing monotone (k, n)-inverters can also be used to construct negation-limited inverters. It is curious that this is true of all known (k, n)-inverters, since no converse of Theorem 4.4 is known to hold.

5. The complexity of the inverter: Lower bounds. In this section, we shall prove a main technical lemma on functions computed at NEGATION gates in negation-limited inverters. These techniques have recently been generalized to obtain results for negation-limited circuits computing symmetric functions [5].

For $x \in \{0, 1\}^n$, let $|x|$ denote the number of ones in x. Note that in all known constructions of negation-limited inverters, the negation gates compute the negations of the bits of $|x|$. We shall see that this must always hold.

LEMMA 5.1. *Let $n = 2^r - 1$. Consider any r-circuit Δ which computes I_n. Label the NEGATION gates $N_1, \ldots, N_r$ in such a way that the input to N_i does not depend*

on the outputs of any of the negation gates $N_{i+1}, \ldots, N_r$ (such a labeling exists since the circuit is a DAG). Let z_i and y_i be the functions computed at the input and output of N_i. Then $z_1 z_2 \ldots z_r$ is the binary representation of $|x|$.

Proof. We proceed by induction on r. The lemma is trivially true for $r = 0$ (i.e., the circuit is monotone). We now suppose that the lemma holds for $r-1$ NEGATION gates.

Let $a \in \{0,1\}^n$. We wish first to show that $z_1(a)$ is the first (i.e., leftmost) bit of $|a|$. We consider the case in which $z_1(a) = 1$ (the proof for the case $z_1(a) = 0$ is analogous), and we wish to show that $|a| \geq (n+1)/2$. Note that simultaneously permuting the inputs and output of Δ according to the same permutation yields another circuit for I_n, so we may assume that a is of the form $1^k 0^{n-k}$. Also, I_{n-k} is obtained from I_n, fixing the first k bits to 1 and ignoring the first k output bits. Since $z_1(a) = 1$, and z_1 is monotone, I_{n-k} is computed by the circuit Δ_k obtained by fixing the first k inputs of Δ to 1 and replacing N_1 by the constant 0. Note that Δ_k is an $(r-1)$-circuit. Therefore, by Markov's theorem, $d(I_{n-k}) \leq 2^{r-1} - 1 = (n-1)/2$. However, it is clear that $d(I_{n-k}) = n - k$. Therefore

$$|a| = k = n - d(I_{n-k}) \geq n - (n-1)/2 = (n+1)/2,$$

as desired.

Now it remains to show that the functions $z_2, \ldots, z_r$ are as specified. Again, we restrict our attention to the case $z_1 = 1$, since the case $z_1 = 0$ is handled similarly. Let $a \in \{0,1\}^n$ such that $z_1(a) = 1$. As above, we assume that a is a string of the form $1^\ell 0^{n-\ell}$. Let $k = (n-1)/2 \leq \ell$. We apply the $r-1$ case of the lemma to the $(n-k)$-input/output inverter obtained from Δ by fixing the first k inputs to 1 and replacing N_1 by the constant 0. The proof is completed by observing that for all $x \in \{0,1\}^{n-k}$, the binary representations of $|x|$ and $|1^k x|$ agree in their $(r-1)$ least significant bits. □

In the sequel, we assume that $z_1, \ldots, z_r$ are as in the lemma. Let N_i be the NEGATION gate corresponding to z_i. We remark that we did not assume that there exists a path from N_{i-1} to N_i for all $2 \leq i \leq r$. (By path we mean a sequence of gates $G_1, \ldots, G_p$ which satisfies the following conditions: $G_1 = N_i$, $G_p = N_{i+1}$, and for any q $(1 \leq q \leq p-1)$, an output of G_q is an input of G_{q+1}.) However, the existence of such paths follows at once from the lemma: if for some i, there did not exist a path from N_{i-1} to N_i, then we could interchange the labels of N_{i-1} and N_i. This is impossible, since by the lemma the labeling of a negation gate is determined by what function is computed at the gate.

LEMMA 5.2. *For any i, $1 \leq i \leq r-1$, in any path from N_i to N_{i+1}, there exists at least one AND gate and one OR gate.*

Proof. The proof is by contradiction. By Lemma 5.1, all four possible values for z_i and z_{i+1} are attained for some input $x \in \{0,1\}^n$. Note that the NEGATION gates are labeled in such a way that no path from N_i to N_{i+1} contains a NEGATION gate other than N_i and N_{i+1}. If the internal nodes along a path from N_i to N_{i+1} consisted entirely of AND (respectively, OR) gates, then the possibility $z_i = 1, z_{i+1} = 1$ (respectively, $z_i = 0, z_{i+1} = 0$) would be ruled out. Therefore, all such paths must include both an AND gate and an OR gate. □

Let L_j $(1 \leq j \leq n)$ be a gate computing $\neg x_j$ in Δ.

LEMMA 5.3. *For all $1 \leq i \leq r$ and $1 \leq j \leq n$, there exists a path from N_i to L_j which does not include any NEGATION gate except for N_i.*

Proof. First, we fix all the outputs of the NEGATION gates $N_1, \ldots, N_r$ in Δ to

0. Then, we obtain an (n, n)-inverter C from Δ. In this case, $C(a) = (0, \ldots, 0)$ for all $a \in \{0,1\}^n$. Let us fix all the inputs of the circuit C to 1.

Next, for some $1 \leq i \leq r$, we change the output of N_i to 1. Then, for some $1 \leq k \leq n-1$, we obtain a (k, n)-inverter by the proof of Lemma 4.4. Since we fixed all the inputs of C to 1, each of the outputs of $L_1, \ldots, L_n$ in C changes from 0 to 1. Thus, for all $1 \leq j \leq n$, there exists a path from N_i to L_j, such that the output of every gate in the path changes from 0 to 1. Since we fixed all the outputs of the NEGATION gates except for N_i to 0, we can conclude that there is no NEGATION gate other than N_i in the path. □

LEMMA 5.4. *For all j, $1 \leq j \leq n$, on any path from N_r to L_j, there exist at least one AND gate and one OR gate.*

Proof. Note that at least one such path exists by the previous lemma. Now we wish to show that any such path contains an AND gate and an OR gate. As above, we note that by Lemma 5.1, each of the four possibilities for the values of z_i, L_j are attained for some input $x \in \{0,1\}^n$. Also, no path from N_r to L_j contains a NEGATION gate other than N_r. Therefore, as in the previous lemma, we see that any path from N_r to L must include at least one AND gate and at least one OR gate (all other possibilities lead to contradictions). □

THEOREM 5.5. *Let $n = 2^r - 1$. Then $C^r(I_n) \geq 5n + 3\log(n+1) - c$.*

Proof. By Lemma 5.1, z_1 is the majority function, computed by a monotone subcircuit, so the number of gates required to compute z_1 is at least $C^m(T^n_{(n+1)/2})$. Since it is known that $C^m(T^n_{n/2}) \geq 4n - c$ where c is a constant (see [10]), we have $C^m(T^n_{(n+1)/2}) \geq 4n - c'$.

From Lemma 5.2, the length of any path from N_1 to N_r is at least $3r - 2$. Note that the gates on such a path are distinct from the gates in the subcircuit computing z_1.

Furthermore, from Lemma 5.3, for all j, $1 \leq j \leq n$, there exists a path from N_r to L_j. So, for all j, $1 \leq j \leq n$, L_j differs from any gate in any path from N_1 to N_r. And the functions computed at $L_1, \ldots, L_n$ are mutually distinct nonmonotone functions (i.e., $\neg x_1, \ldots, \neg x_n$). So $L_1, \ldots, L_n$ are mutually distinct gates, and none of them are used to compute z_1. Hence, we have

$$\begin{aligned} C^r(I_n) &\geq C^m(T^n_{(n+1)/2}) + (3r-2) + n \\ &\geq (4n - c') + (3\log(n+1) - 2) + n \\ &\geq 5n + 3\log(n+1) - c \end{aligned}$$

as desired. □

THEOREM 5.6. *Let $n = 2^r - 1$. Then $D^r(I_n) \geq 4\log(n+1) + c$.*

Proof. From Lemma 5.1, the number of gates used to compute z_1 is at least $C^m(T^n_{(n+1)/2})$. These gates are located below N_1; that is, there exists a path from each of these gates to N_1.

From Lemma 5.2, the length of any path from N_1 to N_r is at least $3r - 2$. Furthermore, from Lemma 5.4, for all j, $1 \leq j \leq n$, the length of any path from N_r to L_j is at least 3. Then, we obtain

$$\begin{aligned} D^l(I_n) &\geq \lceil \log(C^m(T^n_{(n+1)/2}) + 1) \rceil + (3r-2) + 3 - 1 \\ &\geq \lceil \log(4n - c' + 1) \rceil + 3r \\ &\geq \log(c''(n+1)) + 3\log(n+1) \\ &\geq 4\log(n+1) + c \end{aligned}$$

as desired. □

6. Superlinear lower bounds for particular inverters. First, we consider so-called *synchronous circuits* which are circuits with the property that all paths from the inputs to a given gate have the same length (see [17]).

THEOREM 6.1. *Let $n = 2^r + 1$. Then any synchronous negation-limited circuit for V_n has at least $4n\log(n+1) + cn$ gates.*

Proof. From Theorem 5.6, for each i, $1 \le i \le n$, the length of some (and therefore any) path from an input to L_i is at least $4\log(n+1) + c$. In a synchronous circuit computing I_n, the outputs of the gates at level m ($2 \le m \le 4\log(n+1) + c$) are computed using the outputs of the gates at level $m-1$.

On the other hand, V_n can compute 2^n different values, i.e., all binary vectors of length n. Hence, the number of gates at level m is at least $\log 2^n = n$. This completes the proof. □

Next, we show another instance where the size of the negation-limited inverter has a superlinear lower bound. For that purpose, we introduce the following restriction.

Restriction A: Any gate G in the inverter $\mathcal{I}_n$ which satisfies the following two conditions computes some symmetric function:

A1: there exists a path from one of $N_1, \ldots, N_{r-1}$ to G in $\mathcal{I}_n$,

A2: there exists a path from G to N_r in $\mathcal{I}_n$.

THEOREM 6.2. *Let $n = 2^r - 1$. Then any negation-limited inverter Δ which satisfies Restriction A has $\Omega(n \log n)$ gates.*

Proof. Label the NEGATION gates of Δ as in Lemma 5.1. Let z_i be the function which is computed at the input of N_i, for $i = 1, \ldots, r$. By Lemma 5.1, for $x \in \{0,1\}^n$ the string $z_1(x)z_2(x)\ldots z_r(x)$ is the binary representation of $|x|$. This is an increasing function which attains all possible values in the range $0, \ldots, n$.

Let $1 \le k \le n$ be arbitrary. We wish to show that there exists a monotone subcircuit of Δ computing T_n^k. Let w be such that $w01^{r-i}$ is the binary representation of $k-1$, where w is a binary string $w_1 w_2 \ldots w_i$ of length i for some $0 \le i < r$. Then the binary representation of k is $w10^{r-i}$.

Let Δ' be a circuit obtained from Δ by replacing N_j with the constant $\neg w_j$ for each $1 \le j \le i$. For a gate G in Δ, let G' denote the corresponding gate in Δ'. Let g and g' denote the functions computed at gates G and G', respectively. Let $\Omega \subseteq \{0,1\}^n$ denote the set of all binary strings with exactly $k-1$ or k ones. We have:

1. for any gate G in Δ, and for any $x \in \Omega$, $g(x) = g'(x)$;
2. if for all $j > i$, there is no path from N_j to G in Δ, then g' is computed by a monotone subcircuit of Δ';
3. if g is computed by a monotone subcircuit of Δ, then $g = g'$.

Let G be the gate computing z_{i+1}. Note that by (2), g' is monotone, and by (1), $g'(x) = g(x) = T_n^q(x)$ for $x \in \Omega$. Since T_n^k is the unique monotone function which agrees with T_n^k on Ω, we have $g'(x) = T_n^k(x)$ for all $x \in \{0,1\}^n$.

Recall that we are trying to prove that there is a monotone subcircuit of Δ computing T_n^k (from which the theorem follows). So far we have shown the existence of a gate G in Δ such that $g' = T_n^k$. It now suffices to show that for any such gate, either g is computed by a monotone subcircuit of Δ (so, by (3), $g = g'$ and we are done) or there is a gate lower down in the circuit with the same property.

Suppose that g is not computed by a monotone subcircuit of Δ. Then there is a path from some negation gate N_j to G in Δ. This is illustrated in Figure 3.

By Restriction A, we know that g is a symmetric function. Let G_1 and G_2 be the gates whose outputs are the inputs to G, labeled such that a path from N_j to

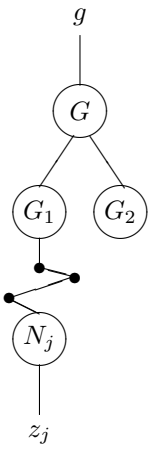


FIG. 3. *A gate G such that G' computes T_n^k. One of G_1, G_2 shares this property.*

G passes through G_1. Therefore g_1 computes a symmetric function. Since g_1' is a monotone function which agrees with g_1 on Ω, either $g_1' = T_n^k$ or g_1' is constant on Ω. In the latter case, since g' is not constant on Ω, we have that either g_1' is identically 1 on Ω and G is an AND gate, or g_1' is identically 0 on Ω and G is an OR gate. So g' and g_2' agree on Ω, and $g_2' = T_n^k$. In either case we have found a gate G_ℓ below G in Δ with $g_\ell' = T_n^k$ as desired. This can be continued until a monotone subcircuit of Δ is found which computes T_n^k.

Since k above was arbitrary, all threshold functions $\{T_n^1, T_n^2, \ldots, T_n^n\}$ are computed by monotone subcircuits of Δ. Thus, from [9], Δ has size $\Omega(n \log n)$. □

Note that, if we do not have the condition A2, certain symmetric functions must be computed at the output gates of Δ_n. But this is impossible. Hence, the condition A2 is necessary although it is not used in the proof of Theorem 6.2.

Since the inverter we construct satisfies Restriction A and has size $O(n \log n)$, this theorem is tight to within a constant factor.

7. Conclusion. In this paper, we have presented new upper and lower bounds on the size of negation-limited inverters. We have also investigated the negation-limited circuit complexity of some boolean functions and have shown relationships between combinational complexity and negation-limited complexity of boolean functions.

It should be investigated whether the methods used in this paper to derive lower bounds on the negation-limited complexity of the inverter can be applied to derive similar bounds on the negation-limited complexity of one-output boolean functions. We have made some progress in this regard for symmetric functions [5].

G. Turán posed the following question: is the size of any $c \log n$ depth inverter using $c \log n$ NEGATION gates superlinear? Though Turán's question remains open, we hope that our work represents a step towards its resolution.

Acknowledgments. The authors would like to express their sincere thanks to Dr. Jaikumar Radhakrishnan at Tata Institute of Fundamental Research for his detailed comments on the previous version of this paper. They also thank Prof. Magnús M. Halldórsson at the University of Iceland, Prof. Akihiro Nozaki at Otsuma Women's University, and Prof. Shigeki Iwata at the University of Electro-Communications for their valuable comments.

REFERENCES

[1] M. Ajtai, J. Komlós, and E. Szemerédi, *An $O(n \log n)$ sorting network*, in Proc. 15th Annual ACM Symposium on the Theory of Computing, ACM, New York, 1983, pp. 1–9.

[2] S. B. Akers, Jr., *On maximum inversion with minimum inverters*, IEEE Trans. Comput., 17 (1968), pp. 134–135.

[3] N. Alon and R. B. Boppana, *The monotone circuit complexity of Boolean functions*, Combinatorica, 7 (1987), pp. 1–22.

[4] K. E. Batcher, *Sorting networks and their applications*, in Proc. 32nd Annual AFIPS Spring Joint Computer Conference, 1968, pp. 307–314.

[5] R. Beals, T. Nishino, and K. Tanaka, *More on the complexity of negation-limited circuits*, in Proc. 27th Annual ACM Symposium on the Theory of Computing, ACM, New York, 1995, pp. 585–595.

[6] S. J. Berkowitz, *On Some Relationships between Monotone and Non-Monotone Circuit Complexity*, Technical Report, University of Toronto, Ontario, Canada, 1982.

[7] R. B. Boppana and M. Sipser, *The complexity of finite functions*, in Handbook of Theoretical Computer Science, Vol. A, J. van Leeuwen, ed., North-Holland, Amsterdam, 1990, pp. 757–804.

[8] M. J. Fischer, *The complexity of negation-limited networks–a brief survey*, in Lecture Notes in Computer Science 33, Springer-Verlag, Berlin, 1974, pp. 71–82.

[9] E. A. Lamagna and J. E. Savage, *Combinational complexity of some monotone functions*, in Proc. 15th Annual IEEE Symposium on Switching and Automata Theory, IEEE, Piscataway, NJ, 1974, pp. 140–144.

[10] D. Long, *The Monotone Circuit Complexity of Threshold Functions*, unpublished manuscript, Oxford University, Oxford, UK, 1986.

[11] A. A. Markov, *On the inversion complexity of a system of functions*, J. Assoc. Comput. Mach., 5 (1958), pp. 331–334.

[12] S. Muroga, *Threshold Logic and Its Applications*, Wiley, New York, 1971.

[13] N. J. Pippenger, *Communication networks*, in Handbook of Theoretical Computer Science, Vol. A, J. van Leeuwen, ed., North-Holland, Amsterdam, 1990, pp. 805–833.

[14] R. Raz and A. Wigderson, *Monotone circuits for matching require linear depth*, J. Assoc. Comput. Mach., 39 (1992), pp. 736–744.

[15] A. A. Razborov, *Lower bounds for the monotone complexity of some Boolean functions*, Soviet Math. Dokl., 31 (1985), pp. 354–357.

[16] M. Santha and C. Wilson, *Limiting negations in constant depth circuits*, SIAM J. Comput., 22 (1993), pp. 294–302.

[17] J. E. Savage, *The Complexity of Computing*, Wiley, New York, 1976.

[18] K. Tanaka and T. Nishino, *On the complexity of negation-limited Boolean networks*, in Proc. 26th Annual ACM Symposium on the Theory of Computing, ACM, New York, 1994, pp. 38–47.

[19] É. Tardos, *The gap between monotone and non-monotone circuit complexity is exponential*, Combinatorica, 7 (1987), pp. 141–142.

[20] L. G. Valiant, *Negation is powerless for Boolean slice functions*, SIAM J. Comput., 15 (1986), pp. 531–535.

[21] I. Wegener, *On the complexity of slice functions*, Theoretical Comput. Sci., 38 (1985), pp. 55–68.

[22] I. Wegener, *The Complexity of Boolean Functions*, Wiley, New York, 1987.

SIAM J. COMPUT.
Vol. 27, No. 5, pp. 1348–1375, October 1998
© 1998 Society for Industrial and Applied Mathematics
009

SIMPLE FAST PARALLEL HASHING BY OBLIVIOUS EXECUTION*

JOSEPH GIL† AND YOSSI MATIAS‡

Abstract. A *hash table* is a representation of a set in a linear size data structure that supports constant-time membership queries. We show how to construct a hash table for any given set of n keys in $O(\lg \lg n)$ parallel time with high probability, using n processors on a weak version of a concurrent-read concurrent-write parallel random access machine (CRCW PRAM). Our algorithm uses a novel approach of hashing by "oblivious execution" based on probabilistic analysis. The algorithm is simple and has the following structure:

1. Partition the input set into buckets by a random polynomial of constant degree.
2. For $t := 1$ to $O(\lg \lg n)$ do
   (a) Allocate M_t memory blocks, each of size K_t.
   (b) Let each bucket select a block at random, and try to injectively map its keys into the block using a random linear function. Buckets that fail carry on to the next iteration.

The crux of the algorithm is a careful a priori selection of the parameters M_t and K_t. The algorithm uses only $O(\lg \lg n)$ random words and can be implemented in a work-efficient manner.

Key words. parallel computation, hashing, data structures, randomization

AMS subject classifications. 68P05, 68P10, 68P15, 68P20, 68Q10, 68Q22

PII. S0097539794291580

1. Introduction.

Let S be a set of n keys drawn from a finite universe U. The *hashing* problem is to construct a function $H : U \to [0, m-1]$ with the following attributes:

Injectiveness. No two keys in S are mapped by H to the same value.

Space efficiency. Both m and the space required to represent H are $O(n)$.

Time efficiency. For every $x \in U$, $H(x)$ can be evaluated in $O(1)$ time by a single processor.

Such a function induces a linear space data structure, a *perfect hash table*, for representing S. This data structure supports membership queries in $O(1)$ time.

This paper presents a simple, fast, and efficient parallel algorithm for the hashing problem. Using n processors, the running time of the algorithm is $O(\lg \lg n)$ with overwhelming probability, and it is superior to previously known algorithms in several respects.

Computational models. As a model of computation we use the concurrent-read concurrent-write parallel random access machine (CRCW PRAM) family (see, e.g., [34]). The members of this family differ by the outcome of the event where more than one processor attempts to write simultaneously into the same shared memory location. The main submodels of CRCW PRAM in descending order of power are the

*Received by the editors April 12, 1994; accepted for publication (in revised form) July 27, 1996; published electronically May 19, 1998. Parts of this research were presented in preliminary form in *Proc.* 2*nd Annual ACM-SIAM Symposium on Discrete Algorithms*, January 1991, and in *Proc.* 21*st International Colloquium on Automata, Languages and Programming*, Lecture Notes in Comput. Sci. 820, Springer, 1994, pp. 239–250.

http://www.siam.org/journals/sicomp/27-5/29158.html

†Department of Computer Science, The Technion—Israel Institute of Technology, Technion City, Haifa 32000, Israel (yogi@cs.technion.ac.il). Part of this research was done while the author was at the Hebrew University and at the University of British Columbia.

‡AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974 (matias@research.att.com). Part of this research was done while the author was at Tel Aviv University and at the University of Maryland Institute for Advanced Computer Studies, and was partially supported by NSF grants CCR-9111348 and CCR-8906949.

PRIORITY [28] in which the lowest-numbered processor succeeds; the ARBITRARY [41] in which one of the processors succeeds, and it is not known in advance which one; the COLLISION$^+$ [8] in which if different values are attempted to be written, a special collision symbol is written in the cell; the COLLISION [14] in which a special collision symbol is written in the cell; the TOLERANT [31] in which the contents of that cell do not change; and, finally, the less standard ROBUST [33] in which if two or more processors attempt to write into the same cell in a given step, then, after this attempt, the cell can contain any value.

1.1. Previous work. Hash tables are fundamental data structures with numerous applications in computer science. They were extensively studied in the literature; see, e.g., [36, 39] for a survey or [40] for a more recent one. Of particular interest are perfect hash tables, in which every membership query is guaranteed to be completed in constant time in the worst case. Perfect hash tables are perhaps even more significant in the parallel context, since the time for executing a batch of queries in parallel is determined by the slowest query.

Fredman, Komlós, and Szemerédi [15] were the first to solve the hashing problem in expected linear time for any universe size and any input set. Their scheme builds a *two-level* hash function: a *level-one* function splits S into subsets ("buckets") whose sizes are distributed in a favorable manner. Then, an injective *level-two* hash function is built for each subset by allocating a private memory block of an appropriate size.

This two-level scheme formed a basis for algorithms for a dynamic version of the hashing problem, also called the *dictionary problem*, in which insertions and deletions may change S dynamically. Such algorithms were given by Dietzfelbinger, Karlin, Mehlhorn, Meyer auf der Heide, Rohnert, and Tarjan [11]; Dietzfelbinger and Meyer auf der Heide [13]; and by Dietzfelbinger, Gil, Matias, and Pippenger [10].

In the parallel setting, Dietzfelbinger and Meyer auf der Heide [12] presented an algorithm for the dictionary problem. For each fixed $\epsilon \geq 0$, n arbitrary dictionary instructions (insert, delete, or lookup) can be executed in $O(n^\epsilon)$ expected time on a $n^{1-\epsilon}$-processor PRIORITY CRCW. Matias and Vishkin [38] presented an algorithm for the hashing problem that runs in $O(\lg n)$ expected time using $O(n/\lg n)$ processors on an ARBITRARY CRCW. This was the fastest parallel hashing algorithm previous to our work. It is based on the two-level scheme and makes extensive use of counting and sorting procedures.

The only known lower bounds for parallel hashing were given by Gil, Meyer auf der Heide, and Wigderson [26]. In their (rather general) model of computation, the required number of parallel steps is $\Omega(\lg^* n)$. They also showed that in a more restricted model, where at most one processor may simultaneously work on a key, parallel hashing time is $\Omega(\lg \lg n)$. They also gave an algorithm which yields a matching upper bound if only function applications are charged and all other operations (e.g., counting and sorting) are free. Our algorithm falls within the realm of the above-mentioned restricted model and matches the $\Omega(\lg \lg n)$ lower bound while charging for all operations on the concrete PRAM model.

1.2. Results. Our main result is that a linear static hash table can be constructed in $O(\lg \lg n)$ time with high probability and $O(n)$ space, using n processors on a CRCW PRAM. Our algorithm has the following properties.

Time optimality. It is the best possible result that does not use processor reallocations, as shown in [26]. Optimal speedup can be achieved with a small penalty in execution time. It is a significant improvement over the $O(\lg n)$-time algorithm of [38].

1. Partition the input set into buckets by a random polynomial of constant degree.
2. For $t := 1$ to $O(\lg \lg n)$ do
   (a) Allocate M_t memory blocks, each of size K_t.
   (b) Let each bucket select a block at random, and try to injectively map its keys into the block using a random linear function; if the same block was selected by another bucket, or if no injective mapping was found, then the bucket carries on to the next iteration.

FIG. 1. *The template for the hashing algorithm.*

Reliability. The time bound $O(\lg \lg n)$ is obeyed with high probability; in contrast, the time bound of the algorithm in [38] is guaranteed only with constant probability.

Simplicity. It is arguably simpler than any other hashing algorithm previously published. (Nevertheless, the analysis is quite involved due to tight tradeoffs between the probabilities of conflicting events.)

Reduced randomness. It is adapted to consume only $O(\lg \lg n)$ random words, compared with $\Omega(n)$ random words that were previously used.

Work optimality. A work-optimal implementation is presented, in which the time-processor product is $O(n)$ and the running time is increased by a factor of $O(\lg^* n)$; it also requires only $O(\lg \lg n)$ random words.

Computational model. If we allow the lookup time to be $O(\lg \lg n)$ as well, then our algorithm can be implemented on the ROBUST CRCW model.

Our results can be summarized in the following theorem.

THEOREM 1.1. *Given a set of n keys drawn from a universe U, the hashing problem can be solved using $O(n)$ space:* (i) *in $O(\lg \lg n)$ time with high probability, using n processors, or* (ii) *in $O(\lg \lg n \lg^* n)$ time and $O(n)$ operations with high probability. The algorithms run on a* CRCW PRAM*, where no reallocation of processors to keys is employed, and use $O(\lg \lg |U| + \lg n \lg \lg n)$ random bits.*

The previous algorithms implementing the two-level scheme, either sequentially or in parallel, are based on grouping the keys according to the buckets to which they belong, and require learning the size of each bucket. Each bucket is then allocated a private memory block whose size is dependent on the bucket size. This approach relies on techniques related to sorting and counting, which require $\Omega(\lg n / \lg \lg n)$ time to be solved by a polynomial number of processors, as implied by the lower bound of Beame and Håstad [4]. This lower bound holds even for randomized algorithms. (More recent results have found other, more involved, ways to circumvent these barriers; cf. [37, 3, 25, 29].)

We circumvent the obstacle of learning buckets sizes for the purpose of appropriate memory allocation by a technique of *oblivious execution* sketched by Figure 1.

The crux of the algorithm is a careful a priori selection of the parameters M_t and K_t. For each iteration t, M_t and K_t depend on the expected number of active buckets and the expected distribution of bucket sizes at iteration t in a way that makes the desired progress possible (or rather, likely).

The execution is oblivious in the following sense: all buckets are treated equally, regardless of their sizes. The algorithm does not make any explicit attempt to estimate the sizes of individual buckets and to allocate memory to buckets based on their sizes,

as is the case in the previous implementations of the two-level scheme. Nor does it attempt to estimate the number of active buckets or the distribution of their sizes.

The selection of the parameters M_t and K_t in iteration t is made according to a priori estimates of the above random variables. These estimates are based on properties of the level-one hash function as well as on inductive assumptions about the behavior of previous iterations.

Remark. The hashing result demonstrates the power of randomness in parallel computation on CRCW machines with memory restricted to linear size. Boppana [6] considered the problem of element distinctness: given n integers, decide whether or not they are all distinct. He showed that solving element distinctness on an n-processor PRIORITY machine with bounded memory requires $\Omega(\lg n/\lg\lg n)$ time. "Bounded memory" means that the memory size is an arbitrary function of n but not of the range of the input values. It is easy to see that if the memory size is bounded by $\Omega(n^2)$ then element distinctness can be solved in $O(1)$ expected time by using hash functions (Fact 2.2). This, however, does not hold for linear size memory. Our parallel hashing algorithm implies that when incorporating randomness, element distinctness can be solved in expected $O(\lg\lg n)$ time using n processors on COLLISION$^+$ (which is weaker than the PRIORITY model) with linear memory size.

1.3. Applications. The perfect hash table data structure is a useful tool for parallel algorithms. Matias and Vishkin [38] proposed using a parallel hashing scheme for space reduction in algorithms in which a large amount of space is required for communication between processors. Such algorithms become space efficient and preserve the number of operations. The penalties are in introducing randomization and in having some increase in time. Using our hashing scheme, the time increase may be substantially smaller.

There are algorithms for which, by using the scheme of [38], the resulting time increase is $O(\lg n)$. By using the new scheme, the time increase is only $O(\lg\lg n \lg^* n)$. This is the case in the construction of *suffix trees* for strings [2, 16] and in the naming assignment procedure for substrings over large alphabets [16].

For other algorithms, the time increase in [38] was $O(\lg\lg n)$ or $O\left((\lg\lg n)^2\right)$, while our algorithm leaves the expected time unchanged. Such is the case in integer sorting over a polynomial range [32] and over a super-polynomial range [5, 38].

More applications are discussed in the conclusion section.

1.4. Outline. The rest of the paper is organized as follows. Preliminary technicalities used in our algorithm and its analysis are given in section 2. The algorithm template is presented in greater detail in section 3. Two different implementations, based on different selections of M_t and K_t, are given in the subsequent sections. Section 4 presents an implementation that does not fully satisfy the statement of Theorem 1.1 but has a relatively simple analysis. An improved implementation of the main algorithm, with a more involved analysis, is presented in section 5. In section 6 we show how to reduce the number of random bits. Section 7 explains how the algorithm can be implemented with an optimal number of operations. The model of computation is discussed in section 8, where we also give a modified algorithm for a weaker model. Section 9 briefly discusses the extension of the hashing problem, in which the input may consist of a multiset. Finally, conclusions are given in section 10.

2. Preliminaries. The following inequalities are standard (see, e.g., [1]).

Markov's inequality. Let ω be a random variable assuming nonnegative values only. Then

$$\mathbf{Prob}(\omega > T) < \mathbf{E}(\omega)/T. \tag{1}$$

Chebyshev's inequality. Let ω be a random variable. Then, for $T > 0$,

$$\mathbf{Prob}(|\omega - \mathbf{E}(\omega)| > T) < \mathbf{Var}(\omega)/T^2. \tag{2}$$

Chernoff's inequality. Let ω be a binomial variable. Then, for $T > 0$,

$$\mathbf{Prob}(|\omega - \mathbf{E}(\omega)| > T) = e^{-\Omega(T^2/\mathbf{E}(\omega))}. \tag{3}$$

Terminology for probabilities. We say that an event occurs with *n-dominant probability* if it occurs with probability $1 - n^{-\Omega(1)}$. We use this terminology as follows. If a polylogarithmic number of events are such that each one of them occurs with n-dominant probability, then their conjunction occurs with n-dominant probability as well. We will therefore usually be satisfied by demonstrating that each algorithmic step succeeds with n-dominant probability.

FACT 2.1. *Let $\omega_1, \ldots, \omega_n$ be pairwise independent binary random variables, and let $\omega = \sum_{1 \le i \le n} \omega_i$. Let $\epsilon > 0$ be constant; let $0 < \xi_1 < (1-\epsilon)\mathbf{E}(\omega)$ and $\xi_2 > (1+\epsilon)\mathbf{E}(\omega)$. Then*

1. *$\omega > \xi_1$ with ξ_1-dominant probability,*
2. *$\omega < \xi_2$ with ξ_2-dominant probability, and*
3. *$(1-\epsilon)\mathbf{E}(\omega) \le \omega \le (1+\epsilon)\mathbf{E}(\omega)$ with $\mathbf{E}(\omega)$-dominant probability.*

Proof. Recall the well-known fact that

$$\begin{aligned} \mathbf{Var}(\omega) &= \sum_{1 \le i \le n} \mathbf{Var}(\omega_i) \text{ since } \omega_i \text{ are pairwise independent} \\ &\le \sum_{1 \le i \le n} \mathbf{E}(\omega_i) \text{ since } 0 \le \omega_i \le 1 \\ &= \mathbf{E}(\omega). \end{aligned} \tag{4}$$

1. By inequality (2)

$$\mathbf{Prob}(\omega > \xi_1) \le \mathbf{Prob}(|\omega - \mathbf{E}(\omega)| > \epsilon\mathbf{E}(\omega)) < 1/\epsilon^2\mathbf{E}(\omega) < 1/\epsilon^2\xi_1.$$

2. If $\xi_2 \le \mathbf{E}(\omega)^2$ then by inequality (2)

$$\mathbf{Prob}(\omega > \xi_2) \le \mathbf{Prob}(|\omega - \mathbf{E}(\omega)| > \epsilon\mathbf{E}(\omega)) < 1/\epsilon^2\mathbf{E}(\omega) < 1/\epsilon^2\sqrt{\xi_2}.$$

If $\xi_2 > \mathbf{E}(\omega)^2$ then by inequality (1)

$$\mathbf{Prob}(\omega > \xi_2) < \mathbf{E}(\omega)/\xi_2 < 1/\sqrt{\xi_2}.$$

3. This follows immediately from the above. □

Hash functions. For the remainder of this section, let $S \subseteq U$ be fixed, $|S| = n$. A hash function $h : U \to [0, m-1]$ splits the set S into buckets; *bucket i* is the subset $\{x \in S \mid h(x) = i\}$ and its size is $s_i = \left|S \cap h^{-1}(i)\right|$ for $0 \le i < m$. An element $x \in S$ *collides* if its bucket is not a singleton. The function is *injective*, or *perfect*, if no element collides. Let

$$B_r = B_r(h) = \sum_{0 \le i < m} \binom{s_i}{r}. \tag{5}$$

A function is injective if and only if $B_2 = 0$, since B_2 is the number of collisions of pairs of keys. More generally, B_r is the number of r-tuples of keys that collide under h.

Polynomial hash functions. Let $U = [0, u-1]$ where u is prime. The class of degree-d polynomial hash functions, $d \geq 1$, mapping U into $[0, m-1]$ is

$$\mathcal{H}_m^d := \left\{ h \;\middle|\; h(x) = \left(\sum_{i=0}^{d} c_i x^i \bmod u \right) \bmod m \text{ for some } c_0, \ldots, c_d \in U \right\}.$$

In the rest of this section we consider the probability space in which h is selected uniformly at random from $\mathcal{H}_m^d$, $d \geq 1$.

The following fact and corollary were shown by Fredman, Komlós, and Szemerédi [15] and before that by Carter and Wegman [7]. (The original proof was only for the case $d = 1$; however, the generalization for $d > 1$ is straightforward.)

FACT 2.2. $\mathbf{E}(B_2) \leq n^2/m$.

COROLLARY 2.3. *The hash function h is injective on S with probability at least $1 - n^2/m$.*

Proof. The function h is injective if and only if $B_2 = 0$. By Fact 2.2 and Markov's inequality, the probability that h is not injective is $\mathbf{Prob}(B_2 \geq 1) \leq n^2/m$. □

The following was shown in [10].

FACT 2.4. *If $d \geq 3$ then $B_2 \leq 2n^2/m$ with n^2/m-dominant probability.*

For $r \geq 0$, let B_r be the random variable which is the rth moment of the sequence of s_i's,

$$B_r = B_r(h) = \sum_{0 \leq i < m} s_i^r. \tag{6}$$

It is easy to see that $B_0 = m$ and $B_1 = n$. Further, it can be shown that if $n = O(m)$ and if h were a completely random function, then B_r is linear in n with high probability for all fixed $r \geq 2$. For polynomial hash functions, Dietzfelbinger et al. [11] proved the following fact.

FACT 2.5. *Let $r \geq 0$ and $m \geq n$. If $d \geq r$ then there exists a constant $\sigma_r > 0$, depending only on r, such that*

$$\mathbf{Prob}(B_r \leq 2^{\sigma_r} \cdot n) \;\geq\; 1/2.$$

Tighter estimates on the distribution of B_r were given in [10].

FACT 2.6. *Let $r \geq 2$. If $d \geq r$ then*

$$\mathbf{E}(B_r) \leq n \sum_{1 \leq j \leq r} \left\{ {r \atop j} \right\} \left(\frac{2n}{m} \right)^{j-1},$$

where $\left\{ {r \atop j} \right\}$ is the Stirling number of the second kind.[1]

FACT 2.7. *Let $\epsilon > 0$ be constant. If $d \geq 2r$ and $m \geq n$ then $B_r \leq (1+\epsilon)\mathbf{E}(B_r)$ with n-dominant probability.*

[1] For $k \geq j \geq 0$, the *Stirling number of the second kind*, $\left\{ {k \atop j} \right\} = \frac{1}{j!} \sum_{i=0}^{k} (-1)^{j-i} \binom{j}{i} i^k$, is the number of ways of partitioning a set of k distinct elements into j nonempty subsets (e.g., [30, Chapter 6]).

Allocation. The bucket selects at random one of the M_t memory blocks. If the same block was selected by another bucket, then the bucket remains active and does not participate in the next step.

Hashing. The bucket selects at random two functions from $\mathcal{H}^1_{K_t}$ and then tries to hash itself into the block separately by each of these functions. If either one of the functions is injective, then its description and the memory address of the block are written in the appropriate cell of array ptr and the bucket becomes inactive. Otherwise, the bucket remains active and carries on to the next iteration.

FIG. 2. *The two steps of an iteration, based on oblivious execution.*

3. A framework for hashing by oblivious execution.

3.1. An algorithm template. The input to the algorithm is a set S of n keys, given in an array. The hashing algorithm works in two stages, which correspond to the two-level hashing scheme of Fredman, Komlós, and Szemerédi [15].

In the first stage a level-one hash function f is chosen. This function is selected at random from the class $\mathcal{H}^d_m$, where d is a sufficiently large constant to be selected in the analysis, and $m = \Theta(n)$. The hash function f partitions the input set into m *buckets*; bucket i, $i = 0, \ldots, m-1$, is the set $S \cap f^{-1}(i)$. The first stage is easily implemented in constant time. The main effort is in the implementation of the second stage, which is described next.

The second level of the hash table is built in the second stage of the algorithm. For each bucket a private memory region, called a *block*, is assigned. The address of the memory block allocated to bucket i is recorded in cell i of a designated array ptr of size m. Also, for each bucket, a level-two function is constructed; this function injectively maps the bucket into its block. The descriptions of the level-two functions are written in ptr.

Let us call a bucket *active* if an appropriate level-two function has not yet been found, and *inactive* otherwise. At the beginning of the stage all buckets are active, and the algorithm terminates when all buckets have become inactive. The second stage consists of $O(\lg \lg n)$ iterations, each executing in constant time. The iterative process rapidly reduces the number of active buckets and the number of active keys.

At each iteration t, a new memory segment is used. This segment is partitioned into M_t blocks of size K_t each, where M_t and K_t will be set in the analysis. Each bucket and each key is associated with one processor. The operation of each active bucket in each iteration is given in Figure 2.

In a few of the last iterations, it may become necessary for an iteration to repeat its body more than once but no more than a constant number of times. The precise conditions and the number of repetitions are given in section 5.

The hash table constructed by the algorithm supports lookup queries in constant time. Given a key x, a search for it begins by reading the cell $\mathsf{ptr}[f(x)]$. The contents of this cell defines the level-two function to be used for x as well as the address of the memory block in which x is stored. The actual offset in the block in which x is stored is given by the injective level-two hash function found in the *Hashing* step above.

3.2. Implementations. The algorithm template described above constitutes a framework for building parallel hashing algorithms. The execution of these algorithms is oblivious in the sense that the iterative process of finding level-two hash functions does not require information about the number or size of active buckets. Successful termination and performance are dependent on the a priori setting of the parameters d, M_t, and K_t. The effectiveness of the allocation step relies on having sufficiently *many* memory blocks; the effectiveness of the hashing step relies on having sufficiently *large* memory blocks. The requirement of keeping the total memory linear imposes a tradeoff between the two parameters. The challenge is in finding a balance between M_t and K_t so as to achieve a desired rate of decay in the number of active buckets. The number of active keys can be deduced from the number of active buckets based on the characteristics of the level-one hash function as determined by d.

We will show two different implementations of the algorithm template, each leading to an analysis of a different nature. The first implementation is given in section 4. There, the parameters are selected in such a way that in each iteration the number of active buckets is expected to decrease by a constant factor. Although each iteration may fail with constant probability, there is a geometrically decreasing series which bounds from above the number of active buckets in each iteration. After $O(\lg \lg n)$ iterations, the expected number of active keys and active buckets becomes $n/(\lg n)^{\Omega(1)}$. The remaining keys are hashed in additional constant time using a different approach, after employing an $O(\lg \lg n)$-time load balancing procedure.

From a technical point of view, the analysis of this implementation imposes relatively modest requirements on the level-one hash function, since it only uses first-moment analysis (i.e., Markov's inequality). Moreover, it only requires a simpler version of the hashing step, in which only one hash function from $\mathcal{H}^1_{K_t}$ is being used. The expected running time is $O(\lg \lg n)$, but this running time is guaranteed only with constant probability, which can be made arbitrarily close to 1.

The second implementation is given in section 5. This implementation is characterized by a doubly exponential rate of decrease[2] in the number of active buckets and keys. After $O(\lg \lg n)$ iterations all keys are hashed without any further processing. This implementation is superior in several other respects: its time performance is with high probability, each key is only handled by its original processor, and it forms a basis for further improvements in reducing the number of random bits.

From a technical point of view, the analysis of this implementation is more subtle and imposes more demanding requirements on the level-one hash function, since it uses second-moment analysis (i.e., Chebyshev's inequality). Achieving a doubly exponential rate of decrease required a more careful selection of parameters.

Together, these implementations demonstrate two different paradigms for fast parallel randomized algorithms, each involving a different flavor of analysis. In the first implementation, one only requires an exponential rate of decrease in the problem size and then relies on reallocation of processors to items. (Subsequent works that use this paradigm and its extensions are mentioned in section 10.) This paradigm is relatively easy to understand and not too difficult to analyze using a framework of probabilistic induction and analysis by expectations. The analysis shows that each iteration succeeds with constant probability and that this implies an overall constant success probability. In contrast, the second implementation shows that each iteration succeeds with n-dominant probability and that this implies an overall n-

[2] A sequence $v_0, v_1, \ldots$ decreases at an *exponential* rate if for all t, $v_t \leq v_0/(1+\epsilon)^t$ for some $\epsilon > 0$; the sequence decreases at a *doubly exponential* rate if for all t, $v_t \leq v_0/2^{(1+\epsilon)^t}$ for some $\epsilon > 0$.

dominant success probability. The analysis is significantly more subtle and relies on more powerful techniques of second moment analysis. The second paradigm consists of a doubly exponential rate of decrease in the problem size and hence does not require any wrapup step.

4. Obtaining exponential decrease. This section presents our first implementation of the algorithm template. Using a rather elementary analysis of expectations, we show that at each iteration the problem size decreases by a constant factor with (only) constant probability. The general framework described in section 4.1 shows that this implies that the problem size decreases at an overall exponential rate.

After $O(\lg\lg n)$ iterations, the number of keys is reduced to $n/(\lg n)^{\Omega(1)}$. A simple load balancing algorithm now allocates $(\lg n)^{\Omega(1)}$ processors to each remaining key. Using the relatively large number of allocated processors, each key is finally hashed in constant time.

4.1. Designing by expectation. Consider an iterative randomized algorithm, in which after each iteration some measure of the problem decreases by a random amount. In a companion paper [21] we showed that at each iteration one can actually assume that in previous iterations the algorithm was not too far from its expected behavior. The paradigm suggested is as follows.

> *Design an iteration to be "successful" with a constant probability under the assumption that at least a constant fraction of the previous iterations were "successful."*

It is justified by the following lemma.

LEMMA 4.1 (probabilistic induction [21]). *Consider an iterative randomized process in which, for all $t \geq 0$, the following holds: iteration $t+1$ succeeds with probability at least $1/2$, provided that among the first t iterations at least $t/4$ were successful. Then, with probability $\Omega(1)$, for every $t > 0$ the number of successful iterations among the first t iterations is at least $t/4$.*

4.2. Parameters setting and analysis. Let the level-one function be taken from $\mathcal{H}_m^{10}$; i.e. set $d = 10$. Let

$$r = 10. \tag{7}$$

Further, set

$$m = 4n. \tag{8}$$

Let

$$K_t = 2^{4+\sigma_{10}/5+t/5}, \tag{9}$$

$$M_t = m\, 2^{4-t/4}, \tag{10}$$

where σ_{10} is as in Fact 2.5.

To simplify the analysis, we allow the parameters K_t and M_t to assume nonintegral values. In actual implementation, they must be rounded up to the nearest integer. This does not increase memory requirements by more than a constant factor; all other performance measures can only be improved.

Memory usage. The memory space used is

$$\sum_t M_t K_t = m 2^{8+\sigma_{10}/5} \sum_t 2^{t/5-t/4} = O(n).$$

LEMMA 4.2. *Let v_t be the number of active buckets at the beginning of iteration t. Then,*

$$\mathbf{Prob}\left(\forall\, t \geq 0 \ : \ \ v_t \leq m2^{-t/4}\right) = \Omega(1).$$

Proof. We assume that the level-one function f satisfies

$$B_{10} \leq 2^{\sigma_{10}} \cdot m. \tag{11}$$

By Fact 2.5, (11) holds with probability at least $1/2$.

The proof is continued by using Lemma 4.1. Iteration t is *successful* if $v_{t+1} \leq v_t/2$. Thus, the number of active buckets after j successful iterations is at most $m2^{-j}$.

The probabilistic inductive hypothesis is that among the first t iterations at least $t/4$ were successful; that is,

$$v_t \leq m2^{-t/4}. \tag{12}$$

The probabilistic inductive step is to show that

$$\mathbf{Prob}\,(v_{t+1} \leq v_t/2) \geq 1/2.$$

In each iteration the parameters K_t and M_t were chosen so as to achieve constant deactivation probability for buckets of size at most

$$\beta_t = \sqrt{K_t/8} = 2^{(1+\sigma_{10}/5+t/5)/2}. \tag{13}$$

We distinguish between the following three types of events, "failures," which may cause a bucket to remain active at the end of an iteration.

(i) *Allocation failure*. The bucket may select a memory block which is also selected by other buckets.

Let $\rho_1(t)$ be the probability that a fixed bucket does not successfully reserve a block in the allocation step. Since there are at most v_t buckets, each selecting at random one of M_t memory blocks, $\rho_1(t) \leq v_t/M_t$. By (12) and (10),

$$\rho_1(t) \leq m2^{-t/4}/m2^{4-t/4} = 1/16.$$

(ii) *Size failure*. The bucket may be too large for the current memory block size. As a result, the probability for it to find a level-two hash function is not high enough.

Let v'_t be the number of buckets at the beginning of iteration t that are larger than β_t. By (11),

$$v'_t \cdot \beta_t^{10} \leq B_{10} \leq 2^{\sigma_{10}} m.$$

Therefore, by (13),

$$v'_t \leq 2^{\sigma_{10}} m/\beta_t^{10} = m2^{\sigma_{10}-5(1+\sigma_{10}/5+t/5)} = m2^{-5-t}. \tag{14}$$

Without loss of generality we assume that if $v_{t+1} \leq v_t/2$ then $v_{t+1} = v_t/2$ (i.e., if more buckets that are needed become inactive, then some of them are still considered as active). Thus, for the purpose of analysis,

$$v_t \geq m2^{-t}. \tag{15}$$

We then have

$$v'_t \leq v_t 2^{-5} < v_t/16.$$

(iii) *Hash failure*. A bucket may fail to find an injective level-two hash function even though it is sufficiently small and it has uniquely selected a block.

Let $\rho_3(t)$ be the probability that a bucket of size at most β_t is not successfully mapped into a block of size K_t in the hashing step. By Corollary 2.3 and (13)

$$\rho_3(t) \leq \beta_t^2/K_t = 1/8.$$

A bucket of size at most β_t that successfully reserves a block of size K_t, and that is successfully mapped into it, becomes inactive. The expected number of active buckets at the beginning of iteration $t+1$ can therefore be bounded by

$$\mathbf{E}\left(v_{t+1}\right) \leq v_t\rho_1(t) + v'_t + v_t\rho_3(t) \leq v_t\left(1/16 + 1/16 + 1/8\right) \leq v_t/4.$$

By Markov's inequality,

$$\mathbf{Prob}\left(v_{t+1} \leq v_t/2\right) \geq 1/2,$$

proving the inductive step. The lemma follows. □

LEMMA 4.3. *Let n_t be the number of active keys at the beginning of iteration t. Then*

$$\mathbf{Prob}\left(\forall\, t \geq 0 \;:\; n_t \leq cn2^{-\alpha t}\right) = \Omega(1)$$

for some constants $c, \alpha > 0$.

Proof. It follows from (11), by using a simple convexity argument, that n_t is maximal when all active buckets at the beginning of iteration t are of the same size q_t. In this case, by (11),

$$v_t \cdot q_t^{10} \leq 2^{\sigma_{10}} \cdot m$$

and

$$n_t = v_t \cdot q_t \leq v_t \cdot \left(\frac{2^{\sigma_{10}} m}{v_t}\right)^{0.1} = (2^{\sigma_{10}} m)^{0.1} v_t^{0.9}.$$

Therefore, by Lemma 4.2, the lemma follows. □

By Lemmas 4.2 and 4.3 we have an exponential decrease in the number of active keys and in the number of active buckets with probability $\Omega(1)$. The number of active keys becomes $n/(\lg n)^c$, for any constant $c > 0$, after $O(\lg\lg n)$ iterations with probability $\Omega(1)$.

4.3. A final stage. After the execution of the second stage with the parameter setting as described above, the number of available resources (memory cells and processors) is a factor of $(\lg n)^{\Omega(1)}$ larger than the number of active keys. This resource redundancy makes it possible to hash the remaining active keys in constant time, as described in the remainder of this section.

All keys that were not hashed in the iterative process will be hashed into an auxiliary hash table of size $O(n)$. Consequently, the implementation of a lookup query will consist of searching the key in both hash tables.

The auxiliary hash table is built using the two-level hashing scheme. A level-one function maps the set of active keys into an array of size n. This function is selected at random from a class of hash functions presented by Dietzfelbinger and Meyer auf der Heide [13, Definition 4.1]. It has the property that with n-dominant probability each bucket is of size smaller than $\lg n$ [13, Theorem 4.6(b')]. For the remainder of this section we assume that this event indeed occurs. (Alternatively, we can use the n^ϵ-universal class of hash functions presented by Siegel [42].)

Each active key is allocated $2 \lg n$ processors, and each active bucket is allocated $4(\lg n)^3$ memory. The allocation is done by mapping the active keys injectively into an array of size $O(n/\lg n)$, and by mapping the indices of buckets injectively into an array of size $O(n/(\lg n)^3)$. These mappings can be done in $O(\lg \lg n)$ time with n-dominant probability by using the simple renaming algorithm from [19].

The remaining steps take constant time. We independently select two $\lg n$ linear hash functions and store them in a designated array. These hash functions will be used by all buckets.

The memory allocated to each bucket is partitioned into two $\lg n$ memory blocks, each of size $2 \lg^2 n$. Each bucket is mapped in parallel into its two $\lg n$ blocks by the two $\lg n$ selected linear hash functions, and each mapping is tested for injectiveness. This is carried out by the two $\lg n$ processors allocated to each key. For each bucket, one of the injective mappings is selected as a level-two function. The selection is made by using the simple "leftmost 1" algorithm of [14].

If for any of the buckets all the mappings are not injective, then the construction of the auxiliary hash table fails.

LEMMA 4.4. *Assume that the number of keys that remain active after the iterative process is at most $n/(\lg n)^3$. Then, the construction of the auxiliary hash table succeeds with n-dominant probability.*

Proof. Recall that each bucket is of size at most $\lg n$. A mapping of a bucket into its memory block of size $2(\lg n)^2$ is injective with probability at least $1/2$ by Corollary 2.3. The probability that a bucket has no injective mapping is therefore at most $1/n^2$. With probability at least $1 - 1/n$, every bucket has at least one injective mapping. □

It is easy to identify failure. If the algorithm fails to terminate within a designated time, it can be restarted. The hash table will be therefore always constructed. Since the overall failure probability is constant, the expected running time is $O(\lg \lg n)$.

5. Obtaining doubly exponential decrease. The implementation of the algorithm template that was presented in the previous section maintains an exponential decrease in the number of active buckets throughout the iterations. This section presents the implementation in which the number of active buckets decreases at a doubly exponential rate.

Intuitively, the stochastic process behind the algorithm template has a potential for achieving doubly exponential rate: if a memory block is sufficiently large in comparison with the bucket size, then the probability of the bucket to remain active is inversely proportional to the size of the memory block (Corollary 2.3). Consider an idealized situation in which this is the case. If at iteration t there are m_t active buckets, each allocated a memory block of size K_t, then at iteration $t + 1$ there will be m_t/K_t active buckets, and each of those could be allocated a memory block of size K_t^2; at iteration $t + 2$ there will be m_t/K_t^3 active buckets, each to be allocated a memory block of size K_t^4, and so on.

In a less idealized setting, some buckets are not deactivated because they are too

large for the current value of K_t. The number of such buckets can be bounded above by using properties of the level-one hash function. It must be guaranteed that the fraction of "large buckets" also decreases at a doubly exponential rate.

The illustrative crude calculation given above assumes that memory can be evenly distributed among the active buckets. To make the doubly exponential rate possible, the failure probability of the allocation step, and hence the ratio m_t/M_t, must also decrease at a doubly exponential rate.

Establishing a bound on the number of "large blocks" and showing that a large fraction of the buckets are allocated memory blocks were also of concern in the previous section. There, however, it was enough to show constant bounds on the probabilities of *allocation failure*, *size failure*, and *hash failure*.

The parameter setting which establishes the balance required for the doubly exponential rate is now presented. Following that is the analysis of the algorithm performance.

5.1. Parameters setting. Let the level-one function be taken from $\mathcal{H}_m^{18}$; i.e., set $d = 18$. Let

$$r = 9. \tag{16}$$

Further, set

$$m = 160n. \tag{17}$$

Let

$$K_t = 2^{a\lambda^t + b_1 t + c_1}, \tag{18}$$

$$M_t = n\, 2^{-a\lambda^t - b_2 t + c_2}, \tag{19}$$

where

$$\lambda = 18/13\,, \quad a = 8/13\,, \quad b_1 = 1/5\,, \quad b_2 = 9/20\,, \quad c_1 = 73/25\,, \quad c_2 = 89/20. \tag{20}$$

5.2. Memory usage.

PROPOSITION 5.1. *The total memory used by the algorithm is $O(n)$.*

Proof. By (17), the memory used in the first stage is $O(n)$. The memory used in an iteration t of the second stage is

$$M_t \cdot K_t = n\, 2^{(b_1 - b_2)t + c_1 + c_2} = n\, 2^{-t/4 + 737/100}. \tag{21}$$

The total memory used by the second stage is therefore at most

$$\sum_{t=0}^{\infty} n\, 2^{-t/4 + 737/100} = \frac{2^{7.37}}{1 - 1/\sqrt[4]{2}} \cdot n = O(n). \qquad \square \tag{22}$$

5.3. Framework for time performance analysis. Let m_t be defined by

$$m_t = n\, 2^{-\lambda^t - b_2 t + 1}. \tag{23}$$

The run-time analysis of the second stage is carried out by showing Lemma 5.2.

LEMMA 5.2. *With n-dominant probability, the number of active buckets in the beginning of iteration t is at most m_t.*

The lemma is proved by induction on t for $t \leq \lg\lg n / \lg\lambda$. The induction base follows from $m_0 = n$ and the fact that there are at most n active buckets.

In the subsequent subsections, we prove the inductive step by deriving estimates on the number of failing buckets in iteration t under the assumption that at the beginning of the iteration there are at most m_t active buckets. Specifically, we show by induction on t that, with n-dominant probability, the number of active buckets at the end of iteration t is at most

$$m_{t+1} = n\, 2^{-\lambda^{t+1} - b_2(t+1)+1}. \tag{24}$$

The bucket may fail to find an injective level-two hash function. In estimating the number of buckets that fail to find an injective level-two function during an iteration we assume that the bucket uniquely selected a memory block and that the bucket size is not too large relatively to the current block size. Accordingly, as in section 4.2, we distinguish between the following three types of events, "failures," which may cause a bucket to remain active at the end of an iteration.

(i) *Allocation failure.* The bucket may select a memory block that is also selected by other buckets.

(ii) *Size failure.* The bucket may be too large for the current memory block size. As a result, the probability for it to find a level-two hash function is not high enough.

(iii) *Hash failure.* A bucket may fail to find a level-two hash function even though it is sufficiently small and it has uniquely selected a block.

We will provide estimates for the number of buckets that remain active due to either of the above reasons: in Lemma 5.5 for case (i), in Lemma 5.6 for case (ii), and in Lemmas 5.7 and 5.8 for case (iii). The estimates are all shown to hold with n-dominant probability. The induction step follows from adding all these estimates.

To wrap up, let $t = \lg\lg n / \lg\lambda$. Then, by (23),

$$m_t = n\, 2^{-\lambda^t - b_2 t + 1} = n\, 2^{-\lg n - b_2 t + 1} < 1.$$

We can therefore infer Proposition 5.3.

PROPOSITION 5.3. *With n-dominant probability, the number of iterations required to deactivate all buckets is at most $\lg\lg n / \lg\lambda$.*

5.4. Failures in uniquely selecting a block.

LEMMA 5.4. *Let ϵ be fixed, $0 < \epsilon < 1/6$, and suppose that either $m_t > M_t^{1/2+\epsilon}$ or $m_t < M_t^{1/2-\epsilon}$. Let ω be the random variable representing the number of buckets that fail to uniquely select a block. Then, $\omega \leq 2m_t^2/M_t$, with M_t-dominant probability.*

Proof. A bucket has a probability of at most m_t/M_t to have other buckets select the memory block it selected. Therefore,

$$\mathbf{E}(\omega) \leq m_t^2/M_t. \tag{25}$$

Further, ω is stochastically smaller than a binomially distributed random variable ϖ obtained by performing m_t independent trials, each with probability m_t/M_t of success. That is to say, $\mathbf{Prob}(\omega \geq \omega_0) \leq \mathbf{Prob}(\varpi \geq \omega_0)$ for all ω_0. Note that $\mathbf{E}(\varpi) = m_t^2/M_t$. If $m_t > M_t^{1/2+\epsilon}$ then

$$\begin{aligned} \mathbf{Prob}\left(\omega > 2m_t^2/M_t\right) &\underset{\text{by (3)}}{\leq} \mathbf{Prob}\left(\varpi > 2m_t^2/M_t\right) \\ &= e^{-\Omega(\mathbf{E}(\varpi))} \\ &= e^{-\Omega(m_t^2/M_t)} \\ &= e^{-\Omega(M_t^{2\epsilon})} \\ &= M_t^{-\Omega(1)}. \end{aligned} \tag{26}$$

Otherwise, $m_t < M_t^{1/2-\epsilon}$ and we are in the situation where $\mathbf{E}(\omega) \ll 1$. Since ω is integer valued and $2m_t^2 > 0$,

$$\begin{array}{rcl} \mathbf{Prob}\left(\omega > 2m_t^2/M_t\right) & \leq & \mathbf{Prob}(\omega \geq 1) \\ & \overset{\text{by (1)}}{\leq} & \mathbf{E}(\omega) \\ & \overset{\text{by (25)}}{\leq} & m_t^2/M_t \\ & < & M_t^{-2\epsilon}. \quad \square \end{array} \tag{27}$$

The setting not covered by the above lemma is $M_t^{1/2-\epsilon} \leq m_t \leq M_t^{1/2+\epsilon}$. This only occurs in a constant number of iterations throughout the algorithm and requires the following special treatment. The body of these iterations is repeated, thus providing a second allocation attempt of buckets that failed to uniquely select a memory block in the first trial.

Let ω_1 and ω_2 be the random variables representing the number of buckets that fail to uniquely select a block in the first and second attempts, respectively,

$$\begin{array}{rcl} \mathbf{Prob}\left(\omega_1 > M_t^{1/2-\epsilon}\right) & \overset{\text{by (1)}}{<} & \mathbf{E}(\omega_1)/M_t^{1/2-\epsilon} \\ & \overset{\text{by (25)}}{=} & m_t^2/M_t M_t^{1/2-\epsilon} \\ & = & m_t^2/M_t^{3/2-\epsilon} \\ & \leq & M_t^{1+2\epsilon}/M_t^{3/2-\epsilon} \\ & = & M_t^{3\epsilon-1/2} \\ & = & M_t^{-\Omega(1)}. \end{array} \tag{28}$$

Therefore, with M_t-dominant probability the second attempt falls within the conditions of equation 27 and hence $\omega_2 = 0$ with M_t-dominant probability.

LEMMA 5.5. *Let $t \leq \lg\lg n/\lg\lambda$. The number of buckets that fail to uniquely select a block is, with n-dominant probability, at most $m_{t+1}/4$.*

Proof. By Lemma 5.4, the number of buckets that fail to uniquely select a memory block is, with M_t-dominant probability, at most

$$\begin{array}{rcl} 2m_t^2/M_t & \overset{\text{by (19),(23)}}{=} & 2n^2\, 2^{-2\lambda^t - 2b_2 t + 2}/n\, 2^{-a\lambda^t - b_2 t + c_2} \\ & = & n\, 2^{(a-2)\lambda^t - b_2 t + 3 - c_2} \\ & \overset{\text{by (20)}}{=} & n\, 2^{(8/13-2)\lambda^t - b_2(t+1) + b_2 + 3 - c_2} \\ & = & n\, 2^{-(18/13)\lambda^t - b_2(t+1) + 9/20 + 3 - 89/20} \\ & \overset{\text{by (20)}}{=} & n\, 2^{-\lambda^{t+1} - b_2(t+1) - 1} \\ & \overset{\text{by (24)}}{=} & m_{t+1}/4. \end{array} \tag{29}$$

The above holds also with n-dominant probability since

$$\begin{array}{rcl} M_t & \overset{\text{by (19)}}{=} & n\, 2^{-a\lambda^t - b_2 t + c_2} \\ & \geq & n\, 2^{-a\lg n - b_2 t + c_2} \\ & = & n^{1-a} 2^{-b_2 \lg\lg n/\lg\lambda + c_2} \\ & \overset{\text{by (20)}}{=} & n^{5/13} \lg n^{-\Omega(1)}. \quad \square \end{array} \tag{30}$$

5.5. Failures in hashing. In considering buckets, which uniquely selected a block which failed to find an injective level-two function, we draw special attention to buckets of size at most

$$\beta_t = \sqrt[4]{K_t/2}. \tag{31}$$

LEMMA 5.6. *The number of buckets larger than β_t is, with n-dominant probability, at most $m_{t+1}/4$.*

Proof. Let $\mu = m/n = 160$. By incorporating into Fact 2.6 the appropriate values for the Stirling numbers of the second kind, we get

$$\mathbf{E}(B_r) \le \left(1 + \frac{510}{\mu} + \frac{12100}{\mu^2} + \frac{62160}{\mu^3} + \frac{111216}{\mu^4} + \frac{84672}{\mu^5} + \frac{29568}{\mu^6} + \frac{4608}{\mu^7} + \frac{256}{\mu^8}\right) n$$
$$\overset{\text{by (17)}}{\le} 4.6756\, n.$$

Therefore, by Fact 2.7, with n-dominant probability

$$B_r \le 6n. \tag{32}$$

From the above and (6) it follows that the number of buckets bigger than β_t is, with n-dominant probability, at most

$$\begin{aligned}
6n/\beta_t^r &\overset{\text{by (31),(16)}}{=} 6n/(K_t/2)^{9/4} \\
&\overset{\text{by (18)}}{=} 6n\, 2^{-9(a\lambda^t + b_1 t + c_1 - 1)/4} \\
&\overset{\text{by (20)}}{=} 6n\, 2^{-(18/13)\lambda^t - (9/20)t - 108/25} \\
&\overset{\text{by (20)}}{=} 6n\, 2^{-\lambda^{t+1} - b_2(t+1) + b_2 - 108/25} \\
&\overset{\text{by (20)}}{=} 6n\, 2^{-\lambda^{t+1} - b_2(t+1) + 9/20 - 108/25} \\
&= 6n\, 2^{-\lambda^{t+1} - b_2(t+1) - 387/100} \\
&= n\, 2^{-\lambda^{t+1} - b_2(t+1) + 1 + (\lg 3 - 387/100)} \\
&\overset{\text{by (24)}}{=} m_{t+1} 2^{\lg 3 - 387/100} \\
&= m_{t+1} 2^{-2.285\ldots} \\
&< m_{t+1}/4. \qquad \square
\end{aligned} \tag{33}$$

The analysis of hashing failures of buckets that are small enough is further split into two cases.

LEMMA 5.7. *Suppose that $m_t/2K_t \ge \sqrt{n}$. Then the number of buckets of size at most β_t that fail in the hashing step of the iteration is, with n-dominant probability, at most $m_{t+1}/4$.*

Proof. Without loss of generality, we may assume that there are exactly m_t active buckets of size at most β_t that participate in the hashing step. When such a bucket is mapped into a memory block of size K_t, the probability of the mapping being noninjective is, by Corollary 2.3, at most $\beta_t^2/K_t = 1/\sqrt{2K_t}$. The probability that the bucket fails in both hashing attempts is therefore at most $1/2K_t$. Let $\tilde{m}_t$ be the total number of such failing buckets. Then, $\mathbf{E}(\tilde{m}_t) \le m_t/2K_t$. By Fact 2.1, with $m_t/2K_t$-dominant probability,

$$
\begin{array}{rcl}
\tilde{m}_t & \leq & 2(m_t/2K_t) \\
& = & m_t/K_t \\
& \overset{\text{by (18),(23)}}{=} & n\,2^{-\lambda^t - b_2 t + 1 - a\lambda^t - b_1 t - c_1} \\
& = & n\,2^{-(1+a)\lambda^t - (b_2+b_1)t + 1 - c_1} \\
& \overset{\text{by (20)}}{\leq} & n\,2^{-(21/13)\lambda^t - b_2(t+1) + 1 + b_2 - c_1} \\
& \overset{\text{by (20)}}{\leq} & n\,2^{-\lambda^{t+1} - b_2(t+1) + 1 + b_2 - c_1} \\
& \overset{\text{by (20),(24)}}{=} & m_{t+1}\,2^{9/20 - 73/25} \\
& < & m_{t+1}/4.
\end{array} \tag{34}
$$

Note that since $m_t/2K_t \geq \sqrt{n}$, the above holds with n-dominant probability and we are done. □

LEMMA 5.8. *Suppose that $m_t/2K_t < \sqrt{n}$. Then, by repeating the hashing step of the iteration a constant number of times, we get $\tilde{m}_t = 0$, with n-dominant probability.*

Proof. We have

$$
n\,2^{-(1+a)\lambda^t - (b_2+b_1)t - c_1} \overset{\text{by (18),(23)}}{=} m_t/2K_t < \sqrt{n}, \tag{35}
$$

and, thus,

$$
2^{(1+a)\lambda^t + (b_2+b_1)t + c_1} > \sqrt{n}. \tag{36}
$$

Therefore,

$$
\begin{array}{rcl}
K_t & \overset{\text{by (18)}}{=} & 2^{a\lambda^t + b_1 t + c_1} \\
& \geq & 2^{\delta\left((1+a)\lambda^t + (b_2+b_1)t + c_1\right)} \\
& \overset{\text{by (36)}}{>} & n^{\delta/2}
\end{array} \tag{37}
$$

for some constant $\delta > 0$. Recall from the proof of Lemma 5.7 that a bucket fails in the hashing step with probability at most $1/2K_t$. By (37), if the iteration body is repeated $\lceil 2/\delta \rceil + 1$ times, the failure probability of each bucket becomes at most $(2K_t)^{-2/\delta - 1} < 2^{-2/\delta}/2K_t n$, and

$$
\mathbf{E}\,(\tilde{m}_t) < m_t 2^{-2/\delta}/2K_t n < 2^{-2/\delta}\sqrt{n}/n = 2^{-2/\delta}/\sqrt{n}.
$$

The lemma follows by Markov's inequality. □

6. Reducing the number of random bits. In this section we show how to reduce the number of random bits used by the hashing algorithm.

The algorithm as described in the previous section consumes $\Theta(n \lg u)$ random bits, where $u = |U|$: the first iteration already uses $\Theta(n \lg u)$ random bits; for each subsequent iteration, the number of random words from U which are used is by at most a constant factor larger than the memory used in that iteration, resulting in a total of $\Theta(n \lg u)$ random bits.

The sequential hashing algorithm of Fredman, Komlós, and Szemerédi [15] can be implemented with only $O(\lg \lg u + \lg n)$ random bits [10]. We show how the parallel hashing algorithm can be implemented with $O(\lg \lg u + \lg n \lg \lg n)$ random bits.

We first show how the algorithm can be modified so as to reduce the number of random bits to $O(\lg u \lg\lg n)$. The first stage requires $O(1)$ random elements from U for the construction of the level-one function and remains unchanged. An iteration t of the second stage required $O(m_t)$ random elements from U; it is modified as follows.

Allocation step. If each bucket independently selects a random memory block, then $O(m_t \lg M_t)$ random bits are consumed. This can be reduced to $O(\lg m)$ by making use of *polynomial hash functions*.

LEMMA 6.1. *Using* $6\lg m$ *random bits, a set* $R \subseteq [0, m-1]$ *of size* m_t *can be mapped in constant time into an array of size* $3M_t$ *such that the number of colliding elements is at most* $2m_t^2/M_t$, *with* M_t*-dominant probability.*

Proof. Let $g_t^a \in \mathcal{H}^3_{2M_t}$ and $g_t^b \in \mathcal{H}^1_{M_t}$ be selected at random. Then the image of a bucket i is defined by

$$g_t(i) = \begin{cases} g_t^a(i) & \text{if } \nexists j \in R,\ j \neq i,\ g_t^a(i) = g_t^a(j), \\ 2M_t + g_t^b(i) & \text{otherwise.} \end{cases} \tag{38}$$

Algorithmically, g_t^a is first applied to all elements and then g_t^b is applied to the elements which collided under g_t^a. The colliding elements of g_t are those which collided both under g_t^a and under g_t^b.

Let R' be the set of elements that collide under g_t^a. Clearly, $|R'| \leq 2B_2(g_t^a)$. Let ϵ be some constant, $0 < \epsilon < 1/6$. Consider the following three cases.

1. $m_t \leq M_t^{1/2-\epsilon}$.
   By Corollary 2.3, $\mathbf{Prob}\,(R' \neq \emptyset) \leq m_t^2/2M_t \leq M_t^{-2\epsilon}/2$.
2. $m_t \geq M_t^{1/2+\epsilon}$.
   It follows from Fact 2.4 that $B_2 \leq 2m_t^2/2M_t = m_t^2/M_t$ with $m_t^2/2M_t$-dominant probability. As $|R'| \leq 2B_2$ and $m_t^2/2M_t \geq M_t^{2\epsilon}/2$ we have that $|R'| \leq 2m_t^2/M_t$ with M_t-dominant probability.
3. $M_t^{1/2-\epsilon} < m_t < M_t^{1/2+\epsilon}$.
   By Fact 2.2, $\mathbf{E}\,(B_2(g_t^a)) \leq m_t^2/2M_t < M_t^{2\epsilon}/2$ and by Markov's inequality,

$$\mathbf{Prob}\left(B_2(g_t^a) > M_t^{1/2-\epsilon}/2\right) \leq M_t^{2\epsilon}/M_t^{1/2-\epsilon} = M_t^{3\epsilon-1/2}.$$

   Therefore, with M_t-dominant probability, $|R'| \leq 2B_2(g_t^a) \leq M_t^{1/2-\epsilon}$, in which case, by Corollary 2.3,

$$\mathbf{Prob}\left(g_t^b \text{ is not injective over } R'\right) \leq |R'|^2/M_t \leq M_t^{-2\epsilon}. \qquad \square$$

Invoking the above procedure for block allocation does not increase the total memory consumption of the algorithm by more than a constant factor.

Hashing step. The implementation of the hashing part of the iteration body using *independent* hash functions for each of the active buckets consumes $O(m_t \lg u)$ random bits. This can be reduced to $O(\lg u)$ by using hash functions which are only *pairwise independent*. This technique and its application in the context of hash functions are essentially due to [9, 10].

The modification to the step is as follows. In each hashing attempt executed during the step, four global parameters $a_0, a_1, b_0, b_1 \in U$ are selected at random by the algorithm. The hash function attempted by a bucket i is

$$h_i(x) := ((c_0(i) + c_1(i)x) \bmod u) \bmod K_t, \tag{39}$$

where

$$c_0(i) = (ia_0 + b_0) \bmod u,$$
$$c_1(i) = (ia_1 + b_1) \bmod u.$$

All hashing attempts of the same bucket are fully independent. Thus, the proof of Lemma 5.8 is unaffected by this modification. Recall that Fact 2.1 assumes only pairwise independence. Since h_i, $i = 0, \ldots, m-1$, are pairwise independent, the proof of Lemma 5.7 remains valid as well.

The above leads to a reduction in the number of random bits used by the algorithm to $O(\lg u \lg\lg n)$.

The number of random bits can be further reduced as follows: employ a preprocessing hashing step in which the input set S is injectively mapped into the range $[0, n^3 - 1]$. This is done by applying a hash function π selected from an appropriate class to map the universe U into this range. Then the algorithm described above is used to build a hash table for the set $\pi(S)$. A lookup of a key x is done by searching for $\pi(x)$ in this hash table.

A class of hash functions that is appropriate for this universe reduction application was described in [10]. This class has the following properties.

1. A selection of a random function π from the class requires $O(\lg\lg u + \lg n)$ random bits.
2. A selection can be made in constant time by a single processor.
3. The function π is injective over S with n-dominant probability.
4. Computing $\pi(x)$ for any $x \in U$ can be done in constant time.

This preprocessing is tantamount to a reduction in the size of the universe, after which application of the algorithm requires only $O(\lg n \lg\lg n)$ bits. The total number of random bits used is therefore

$$O(\lg\lg u + \lg n \lg\lg n).$$

We note that a lookup step is essentially the same as before, except that when looking for an element x in bucket i (i.e., $i = f(x)$), locations $g_t^a(i)$ and $g_t^b(i)$ should both be checked.

7. Obtaining optimal speedup. The description of the algorithm in section 3 assumed that the number of processors is n; thus the time-processor product is $O(n \lg\lg n)$. Our objective in this section is a work-optimal implementation where this product is $O(n)$ and p, the number of processors, is maximized.

When $p < n$, the key array and the bucket array are divided into p *sectors*, one per processor. A parallel step of the algorithm is executed by having each processor traverse its sector and execute the tasks included in it.

A key is *active* if its bucket is active. Let n_t be the number of active keys at the beginning of iteration t. Assume that the implemented algorithm has reached the point where $n_t = O(n/\lg\lg n)$. Further assume that these active elements are gathered in an array of size $O(n/\lg\lg n)$. Then, applying the nonoptimal algorithm of section 3 with $p \le n/\lg\lg n$, and each processor being responsible for $n/p \lg\lg n$ problem instances, gives a running time of

$$O\left(\frac{n}{p \lg\lg n} \lg\lg\left(\frac{n}{\lg\lg n}\right)\right) = O\left(n/p\right),$$

which is work optimal.

We first show that the problem size is reduced sufficiently for the application of the nonoptimal algorithm after $O(\lg\lg\lg\lg n)$ iterations.

LEMMA 7.1. *There exists* $t_0 = O(\lg\lg\lg\lg n)$ *such that* $n_{t_0} = O(n/\lg\lg n)$ *with* n*-dominant probability.*

Proof. The number of active buckets decreases at a doubly exponential rate as can be seen from Lemma 5.2. To see that the number of keys decreases at a doubly exponential rate as well, we show that with n-dominant probability

$$n_t \leq 1.23n2^{-2a\lambda^{t-1}-2b_1t+8/9}. \tag{40}$$

Inequality (32), $B_r \leq 6n$, clearly holds when the summation is over active buckets only. By a convexity argument, the total number of keys in active buckets are maximized when all active buckets are of equal size. The number of active buckets are bounded from above by m_t. Therefore,

$$n_t \leq (6n)^{1/r} m_t^{1-1/r}. \tag{41}$$

Inequality (40) is obtained from (41) by substituting the definition (23) of m_t in (23) and then substituting numerical values for the parameters using (16) and (20).

The lemma follows by choosing an appropriate value for t_0 with respect to (23) and (40). □

It remains to exhibit a work-efficient implementation of the first t_0 steps of the algorithm. This implementation outputs the active elements gathered in an array of size $O(n/\lg\lg n)$. The rest of this section is dedicated to the description of this implementation.

As the algorithm progresses, the number of active keys and the number of active buckets decrease. However, the decrease in the number of active elements in different sectors is not necessarily identical. The time of implementing one parallel step is proportional to the number of active elements in the largest sector. It is therefore crucial to occasionally balance the number of active elements among different sectors in order to obtain work efficiency.

Let the *load* of a sector be the number of active elements (tasks) in it. A *load balancing* algorithm takes as input a set of tasks arbitrarily distributed among p sectors; using p processors it redistributes this set so that the load of each sector is greater than the average load by at most a constant factor. Suppose that we have a load balancing algorithm whose running time, using p processors, is $T_{\text{lb}}(p)$ with n-dominant probability. If load balancing is applied after step t, then the size of each sector is $O(n_t/p)$.

We describe a simple work-optimal implementation in which load balancing is applied after each of the first t_0 parallel steps. A parallel step t executes in time which is in the order of

$$\frac{n_t}{p} + T_{\text{lb}}(p).$$

The total time of this implementation is in the order of

$$\sum_{t=1}^{t_0} \left(\frac{n_t}{p} + T_{\text{lb}}(p) \right).$$

Since n_t decreases at least at an exponential rate, the total time is in the order of

$$\frac{n}{p} + t_0\, T_{\text{lb}}(p),$$

which is $O(n/p)$ for

$$p = O\left(\frac{n}{T_{\text{lb}}(p)\lg\lg\lg\lg n}\right).$$

Using the load balancing algorithm of [19] which runs in $T_{\text{lb}}(p) = O(\lg\lg p)$ time, we conclude that with n-dominant probability the running time on a p-processor machine is

$$O(n/p + \lg\lg p \lg\lg\lg\lg n).$$

The load balancing algorithm applied consumes $O(p\lg\lg p)$ random bits. All these bits are used in a random mapping step which is very similar to the allocation step of the hashing algorithm. Thus, by an approach similar to the mapping procedure in Lemma 6.1 it may be established that the number of random bits in the load balancing algorithm can be reduced to $O(\lg p \lg\lg p)$.

We finally remark that using load balancing in a more efficient way, as described in [22], yields a faster work-efficient implementation. This simple technique is based on carefully choosing the appropriate times for invoking the load balancing procedure; it applies to any algorithm in which the problem size has an exponential rate of decrease, and it hence applies to the implementation of section 4 as well. In such an implementation the load balancing algorithm is only used $O(\lg^* n)$ times, resulting in a parallel hashing algorithm that takes $O(n/p + \lg\lg n \lg^* n)$ time with n-dominant probability.

8. Model of computation. In this section we give closer attention to the details of the implementation on a PRAM, and we study the type of concurrent memory access required by our algorithm. We first present an implementation on COLLISION and its extension to the weaker TOLERANT model. We proceed by presenting an implementation on the even weaker ROBUST model. The hash table constructed in this implementation only supports searches in $O(\lg\lg n)$ time. Finally, we examine the concurrent read capability needed by the implementations.

8.1. Implementation on COLLISION and on TOLERANT. We describe an implementation on COLLISION. This implementation is also valid for TOLERANT, since each step of COLLISION can be simulated in constant time on TOLERANT provided that, as it is the case here, only linear memory is used [31].

Initialization. The selection of the level-one hash function is done by a single processor. Since the level-one function is a polynomial of a constant degree, its selection can be done by a single processor and can be read by all processors in constant time, using a singe memory cell of $\lceil \max\{\lg\lg u, \lg n\} \rceil$ bits. No concurrent write operation is required for the implementation of this stage.

Bucket representatives. The algorithm template assumes that each bucket can act as a single entity for some operations, e.g., selecting a random block and selecting a random hash function. Since usually several keys belong to the same bucket, it is necessary to coordinate the actions of the processors allocated to these keys. A simple way of doing so is based on the fact that there are only linearly many buckets and that a bucket is uniquely indexed by the value of f, the level-one hash function, on its members. A processor whose index is determined by the bucket index acts as the *bucket representative* and performs the actions prescribed by the algorithm to the bucket.

Allocation and hashing steps. A processor representing an active bucket selects a memory block and a level-two hash function and records these selections in a designated cell. All processors with keys in that bucket then read that cell and use the selected block in the hashing step. Each participating processor (whose key belongs in an active bucket) writes its key in the cell determined by its level-two hash function and examines the cell contents to see if the write operation was successful. A processor for which the write failed will then attempt to write its key to position i of array ptr, where i is the number of the bucket this processor belongs to. Processors belonging to bucket i can then learn if the level-two function selected for their bucket is injective by reading the content of ptr$[i]$. A change in value or a collision symbol indicates noninjectiveness. To complete the process, the array ptr is restored for the next hashing attempt. This restoration can be done in constant time since this array is of linear size.

In summary we have Proposition 8.1.

PROPOSITION 8.1. *The algorithms of Theorem* 1.1 *can also be implemented on the* TOLERANT *model.*

8.2. Implementation on ROBUST. We now describe an implementation that, at the expense of slowing down the lookup operation, makes no assumption about the result of a concurrent write into a cell. Specifically, we present an implementation on the ROBUST model for which a lookup query may take $O(\lg \lg n)$ time and, for keys in the table, the expected time is $O(1)$.

The difficulty with the ROBUST model is in letting all processors in a bucket know whether the level-two hash function of their bucket is injective or not. The main idea in the modified implementation is in allowing iterations to proceed without determining whether level-two hash functions are injective or not; whenever a key is written into a memory cell in the hashing step it is deactivated, and its bucket size decreases. The modified algorithm performs at least as well as the implementation in which a bucket is deactivated only if all of its keys are mapped injectively. The total memory used by the modified algorithm and the size of the representation of the hash table do not change.

Allocation step. We first note that the algorithm can be carried out without using bucket representatives at all. Allocation of memory blocks is done using hash functions, as in Lemma 6.1; each processor can individually compute the index of its memory block by evaluating the function g_t. This function is selected by a designated processor and its representation ($6 \lg m$ bits) is read in constant time by all processors.

We further modify the algorithm, so that the hashing step is carried out by *all* active buckets. That is, even buckets that collided in the allocation step will participate in the hashing step. This modification can only serve to improve the performance of the algorithm, since even while sharing a block with another bucket the probability that a bucket finds an injective function into that block is not zero. This modification eliminates the concurrent memory access needed for detecting failures in the allocation step.

Hashing step. The selection of a level-two hash function is done as in the hashing step described in section 6. As can be seen from (39), only four global parameters should be selected and made available to all processors; this can be done in constant time.

It remains to eliminate the concurrent memory access required for determining if the level-two function of any single bucket was injective. Whenever a key is successfully hashed by this function, it is deactivated even if other keys in the same bucket

Let x be an active key in a bucket $i = f(x)$. The processor assigned to x executes the following steps.

Allocation. Compute $g_t(i)$, the index of the memory block selected for the bucket containing x, where g_t is defined by (38).

Hashing. Determine h_i, the level-two hash function selected by the bucket of x, where h_i is defined by (39). Write x into cell $h_i(x)$ in memory block $g_t(i)$ and read the contents of that cell; if x was written then the key x becomes inactive.

FIG. 3. *Implementation of iteration t in the hashing algorithm on* ROBUST.

were not successfully hashed. Thus, keys of the same bucket may be stored in the hash table using different level-two hash functions.

The two steps of an iteration in the hashing algorithm are summarized in Figure 3.

Lookup algorithm. The search for a key x is done as follows. Let $i = f(x)$; for $t = 1, 2, \ldots$ read position $h_i(x)$ in the memory block $g_t(i)$ in the appropriate array. (All random bits that were used in the hash table construction algorithm are assumed to be recorded and available.) The search is terminated when either x is found or when t exceeds the number of iterations in the construction algorithm.

The lookup algorithm requires $O(\lg \lg n)$ iterations in the worst case. However, for any key $x \in S$ the expected lookup time (over all the random selections made by the hashing algorithm) is $O(1)$.

An alternative simplified implementation. Curiously, the sequence of modifications to the algorithm described in this section has lead to a *one-level* hashing scheme, i.e., to the elimination of indirect addressing. To see this, we observe that at iteration t an active key x is written into a memory cell ${g_t}^*(x)$, where the function ${g_t}^*(x)$ is dependent only on n and on the random selections made by the algorithm, but not on the input (beyond the slight dependency of g_t on detecting collisions in the input, which does not imply indirect addressing). An even simpler implementation of a one-level hashing algorithm is outlined next.

At each iteration t, a new array T_t of size $3M_t$ is used, where M_t is as defined in (19). In addition, a function g_t as defined in (38) is selected at random. A processor representing an active key x in the iteration tries to write x into $T_t[g_t^a(x)]$ and then reads this cell. If x is successfully written in $T_t[g_t^a(x)]$ then x is deactivated (and $g_t(x) = g_t^a(x)$). Otherwise, this processor tries to write x into $T_t[g_t^b(x)]$ and then reads this cell. If x is successfully written in $T_t[2M_t + g_t^b(x)]$ then x is deactivated (and $g_t(x) = 2M_t + g_t^b(x)$). Otherwise, x remains active and the processor representing it carries on to the next iteration.

To see that the algorithm terminates in $O(\lg \lg n)$ iterations with n-dominant probability, we observe that the operation on keys in each iteration is the same as the operation on buckets in the allocation step of section 6. Therefore, the analysis of section 6 can be reused, substituting keys for buckets (and ignoring failures in the hashing step of the two-level algorithm). The hash table consists of the collection of the arrays $T_1, T_2, \ldots$, and, as can be easily verified, is of linear size. A lookup query for a given key x is executed in $O(\lg \lg n)$ time by reading $T_t[g_t^a(x)]$ and then (if necessary) $T_t[2M_t + g_t^b(x)]$ for $t = 1, 2, \ldots$.

We note that it is difficult to detect termination on the ROBUST in $o(\lg n)$ time. Therefore, the algorithm described here is of a Monte Carlo type.

8.3. Minimizing concurrent read requirements. The algorithms for construction of the hash table on TOLERANT can be modified to use concurrent read from a single cell only. By allowing a preprocessing stage of $O(\lg n)$ time, concurrent read can be eliminated, implying that the ERCW model is sufficient. With these modifications, parallel lookups still require concurrent read, and their execution time increases to $O(\lg \lg n)$ in the worst case. Nevertheless, the expected time for lookup of any single key $x \in S$ is $O(1)$.

There are two types of concurrent-read operations required by the modified algorithm. First, the sequence of $O(\lg \lg n)$ functions ${g_t}^*$ (or, alternatively, g_t in the simplified implementation) must be agreed upon by all processors. Since each of these functions is represented by $O(\lg u)$ bits, its selection can be broadcasted at the beginning of the iteration through the concurrent-read cell.

The single cell concurrent-read requirement for broadcasting can be eliminated by adding an $O(\lg n)$-time preprocessing step for the broadcasting. (This is just a special case of simulating CRCW PRAM by EREW PRAM.)

The other kind of concurrent-read operation occurs when processors read a memory cell to verify that their hashing into that cell has succeeded. This operation can be replaced by the following procedure. For each memory cell, there is a processor standing by. Whenever a pair $\langle x, j \rangle$ is written into a cell, the processor assigned to that cell sends an acknowledgment to processor j by writing into a memory cell j in a designated array.

The lookup algorithm requires concurrent-read capabilities. In this sense, the lookup operation is more demanding than the construction of the hash table. A similar phenomenon was observed by Karp, Luby, and Meyer auf der Heide [35] in the context of simulating a random access machine on a distributed memory machine. The main challenge in the design of their (parallel-hashing-based) simulation algorithm was the execution of the read step. Congestions during the execution of the write step were resolved by attempting to write in several locations and using the first location for which the write succeeded. It is more difficult to resolve read congestions since the cells in which values were stored are already determined. Indeed, the read operation constitutes the main run-time bottleneck in their algorithm.

9. Hashing of multisets. We conclude the technical discussion by briefly considering a variation of the hashing problem in which the input is a multiset rather than a set of distinct keys. (In this case, the number of keys in the resulting table are equal to the number of distinct keys in the input.) We first note that the analyses of the exponential and doubly exponential rate of decrease in the problem size is not affected by the possibility of multiple occurrences of the same key. This is a result of relying on estimates of the number of active *buckets* rather than on the number of active keys. The number of distinct keys—not the number of keys—determine the probability of a bucket to find an injective function.

A predictable decrease in the number of active *keys* is essential for obtaining an optimal speedup algorithm. Unfortunately, the analysis in section 7 with regard to the implementation of section 5 does not hold. To understand the difficulty, consider the case where a substantial fraction of the input consists of copies of the same key. Then with nonnegligible probability this key may belong to a large bucket. The probability that this bucket deactivates in the first few iterations, in which the memory blocks are not sufficiently large, is too small to allow global decrease in the number of keys

with high probability. Consequently, the rapid decrease in the number of buckets may not be accompanied by a similar decrease in the number of keys.

In contrast, the nature of the analysis in section 4 makes this analysis susceptible to an easy extension to multiple keys, which leads to an optimal speedup algorithm, albeit with expected performance only. Using the probabilistic induction lemma all that is required is to show that each copy of an active key has a constant positive probability of deactivation at each iteration. Since the analysis is based on expectations only, there are no concerns regarding correlations between copies of the same key or dependencies between different iterations. The details are left to the reader.

We also note that the model of computation required for a multiset is COLLISION$^+$, since it must be possible to distinguish between the case of multiple copies of the same key being written into a memory cell and the case where distinct keys are written. Also, the extensions of the hashing algorithms which only require concurrent read from a single memory cell can be used for hashing with multiset input, but then a COLLISION$^+$ model, as opposed to ROBUST, must be assumed.

We finally observe that the hashing problem with a multiset as input can be reduced into the ordinary hashing problem (in which the input consists of a set) by a procedure known as *leaders election*. This procedure selects a single representative from among all processors which share a value. By using a randomized $O(\lg\lg n)$-time, linear-work leaders election algorithm which runs on TOLERANT [23] we have Theorem 9.1.

THEOREM 9.1. *Given a multiset of n keys drawn from a universe U, the hashing problem can be solved using $O(n)$ space:* (i) *in $O(\lg\lg n)$ time with high probability, using n processors, or* (ii) *in $O(\lg\lg n \lg^* n)$ time and $O(n)$ operations with high probability. The algorithms run on* TOLERANT.

Conversely, note that any hashing algorithm, when run on ARBITRARY, solves the leaders election problem. In particular, the simple one-level hashing algorithm for ROBUST, when implemented on ARBITRARY with a multiset as input, gives a simple leaders election algorithm which uses only $O(n)$ space.

Consider now another variant of the multiset hashing problem in which a data record is associated with each key. The natural semantics of this problem is that multiple copies of the same key can be inserted into the hash table only if their data records are identical. Processors representing copies of a key with conflicting data records should terminate the computation with an error code. The COLLISION$^+$ model makes it easy enough to extend the implementations discussed above to accommodate this variant.

A more sophisticated semantics, in which the data records should be consolidated, requires a different treatment, e.g., by applying an integer sorting algorithm on the hashed keys (see [38]).

10. Conclusions. We presented a novel technique of hashing by oblivious execution. By using this technique, algorithms for constructing a perfect hash table which are fast, simple, and efficient, were made possible. The running time obtained is best possible in a model in which keys are only handled in their original processors.

The number of random bits consumed by the algorithm is $\Theta(\lg\lg u + \lg n \lg\lg n)$. An open question is to close the gap between this number and the $\Theta(\lg\lg u + \lg n)$ random bits that are consumed in the sequential hashing algorithm of [10].

The program executed by each processor is extremely simple. Indeed, the only coordination between processors is in computing the AND function, when testing for injectiveness. In the implementation on the ROBUST model, even this coordination is eliminated.

The large constants hidden under the O-notation in the analysis may render the described implementations still far from being practical. We believe that the constants can be substantially improved without compromising the simplicity of the algorithm by a more careful tuning of the parameters and by tightened analysis. This may be an interesting subject of a separate research.

The usefulness of the oblivious execution approach presented in this paper is not limited to the hashing problem alone. We have adopted it in [23] for simulations among submodels of the CRCW PRAM. As in the hashing algorithm, keys are partitioned into subsets. However, this partition is arbitrary and given in the input, and for each subset the maximum key must be computed.

Subsequent work. The oblivious execution technique for hashing from section 3 and its implementation from section 4 were presented in preliminary form in [20]. Subsequently, our oblivious execution technique was used several times to obtain improvements in running time of parallel hashing algorithms: Matias and Vishkin [37] gave an $O(\lg^* n \lg\lg^* n)$ expected-time algorithm; Gil, Matias, and Vishkin [25] gave a tighter failure probability analysis for the algorithm in [37], yielding $O(\lg^* n)$ time with high probability; a similar improvement (from $O(\lg^* n \lg\lg^* n)$ expected time to $O(\lg^* n)$ time with high probability) was described independently by Bast and Hagerup [3].

An $O(\lg^* n)$-time hashing algorithm is used as a building block in a parallel dictionary algorithm presented in [25]. (A parallel dictionary algorithm supports in parallel batches of operations *insert*, *delete*, and *lookup*.) The oblivious execution technique has an important role in the implementation of insertions into the dictionary. The dictionary algorithm runs in $O(\lg^* n)$ time with high probability, improving the $O(n^\epsilon)$-time dictionary algorithm of Dietzfelbinger and Meyer auf der Heide [12]. The dictionary algorithm can be used to obtain a space-efficient implementation of any parallel algorithm, at the cost of a slowdown of at most $O(\lg^* n)$ time with high probability.

The above hashing algorithms use the log-star paradigm of [37], relying extensively on processor reallocation, and are not as simple as the algorithm presented in this paper. Moreover, they require a substantially larger number of random bits.

Karp, Luby, and Meyer auf der Heide [35] presented an efficient simulation of a PRAM on a distributed memory machine in the doubly logarithmic time level, improving over previous simulations in the logarithmic time level. The use of a fast parallel hashing algorithm is essential in their result; the algorithm presented here is sufficient to obtain it.

Goldberg, Jerrum, Leighton, and Rao [27] used techniques from this paper to obtain an $O(h + \lg\lg n)$ randomized algorithm for the h-relation problem on the optical communication parallel computer model.

Gibbons, Matias, and Ramachandran [17] adapted the algorithm presented here to obtain a low-contention parallel hashing algorithm for the QRQW PRAM model [18]; this implies an efficient hashing algorithm on Valiant's BSP model and hence on hypercube-type noncombining networks [43].

Acknowledgments. We thank Martin Dietzfelbinger and Faith E. Fich for providing helpful comments. We also wish to thank Uzi Vishkin and Avi Wigderson for early discussions. Part of this research was done during visits of the first author to AT&T Bell Laboratories, and of the second author to the University of British Columbia. We would like to thank these institutions for supporting these visits. Many valuable comments made by two anonymous referees are gratefully acknowledged.

REFERENCES

[1] N. Alon and J. H. Spencer, *The Probabilistic Method*, John Wiley & Sons, Inc., New York, 1991.

[2] A. Apostolico, C. Iliopoulos, G. M. Landau, B. Schieber, and U. Vishkin, *Parallel construction of a suffix tree*, Algorithmica, 3 (1988), pp. 347–365.

[3] H. Bast and T. Hagerup, *Fast and reliable parallel hashing*, in 3rd Annual ACM Symposium on Parallel Algorithms and Architectures, July 1991, pp. 50–61.

[4] P. Beame and J. Håstad, *Optimal bounds for decision problems on the CRCW PRAM*, J. Assoc. Comput. Mach., 36 (1989), pp. 643–670.

[5] P. C. Bhatt, K. Diks, T. Hagerup, V. C. Prasad, T. Radzik, and S. Saxena, *Improved deterministic parallel integer sorting*, Inform. and Comput., 94 (1991), pp. 29–47.

[6] R. B. Boppana, *Optimal separations between concurrent-write parallel machines*, in Proc. 21st Annual ACM Symposium on Theory of Computing, 1989, pp. 320–326.

[7] L. J. Carter and M. N. Wegman, *Universal classes of hash functions*, J. Comput. System Sci., 18 (1979), pp. 143–154.

[8] B. S. Chlebus, K. Diks, T. Hagerup, and T. Radzik, *New simulations between CRCW PRAMs*, in Proc. 7th International Conference on Fundamentals of Computation Theory, Lecture Notes in Comput. Sci. LNCS 380, Springer, New York, 1989, pp. 95–104.

[9] B. Chor and O. Goldreich, *On the power of two-point based sampling*, J. Complexity, 5 (1989), pp. 96–106.

[10] M. Dietzfelbinger, J. Gil, Y. Matias, and N. Pippenger, *Polynomial hash functions are reliable*, in Proc. 19th International Colloquium on Automata Languages and Programming, Lecture Notes in Comput. Sci. 623, Springer, New York, July 1992, pp. 235–246.

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

[12] M. Dietzfelbinger and F. Meyer auf der Heide, *An optimal parallel dictionary*, in 1st Annual ACM Symposium on Parallel Algorithms and Architectures, 1989, pp. 360–368.

[13] M. Dietzfelbinger and F. Meyer auf der Heide, *A new universal class of hash functions and dynamic hashing in real time*, in Proc. 17th International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 443, Springer, New York, 1990, pp. 6–19.

[14] F. E. Fich, P. L. Ragde, and A. Wigderson, *Relations between concurrent-write models of parallel computation*, SIAM J. Comput., 17 (1988), pp. 606–627.

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

[16] Z. Galil and R. Giancarlo, *Data structures and algorithms for approximate string matching*, J. Complexity, 4 (1988), pp. 33–72.

[17] P. B. Gibbons, Y. Matias, and V. L. Ramachandran, *Efficient low-contention parallel algorithms*, in 6th Annual ACM Symposium on Parallel Algorithms and Architectures, June 1994, pp. 236–247.

[18] P. B. Gibbons, Y. Matias, and V. L. Ramachandran, *The QRQW PRAM: Accounting for contention in parallel algorithms*, in Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms, Jan. 1994, pp. 638–648.

[19] J. Gil, *Fast load balancing on a PRAM*, in Proc. 3rd IEEE Symposium on Parallel and Distributed Computing, Dec. 1991, pp. 10–17.

[20] J. Gil and Y. Matias, *Fast hashing on a PRAM—Designing by expectation*, in Proc. 2nd Annual ACM-SIAM Symposium on Discrete Algorithms, Jan. 1991, pp. 271–280.

[21] J. Gil and Y. Matias, *Designing algorithms by expectations*, Inform. Process. Lett., 51 (1994), pp. 31–34.

[22] J. Gil and Y. Matias, *An effective load balancing policy for geometric decaying algorithms*, J. Parallel and Distributed Computing, 36 (1996), pp. 185–188.

[23] J. Gil and Y. Matias, *Fast and efficient simulations among CRCW PRAMs*, J. Parallel and Distributed Computing, 23 (1994), pp. 135–148.

[24] J. Gil and Y. Matias, *Simple fast parallel hashing*, in Proc. 21st International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 820, Springer, New York, 1994, pp. 239–250.

[25] J. Gil, Y. Matias, and U. Vishkin, *Towards a theory of nearly constant time parallel algorithms*, in Proc. 32nd IEEE Annual Symposium on Foundation of Computer Science, Oct. 1991, pp. 698–710.

[26] J. Gil, F. Meyer auf der Heide, and A. Wigderson, *The tree model for hashing: Lower and upper bounds*, SIAM J. Comput., 25 (1996), pp. 936–955.

[27] L. A. Goldberg, M. Jerrum, F. T. Leighton, and S. B. Rao, *Doubly logarithmic communication algorithms for optical communication parallel computers*, in 5th Annual ACM Symposium on Parallel Algorithms and Architectures, 1993, pp. 300–309.
[28] L. M. Goldschlager, *A universal interconnection pattern for parallel computers*, J. Assoc. Comput. Mach., 29 (1982), pp. 1073–1086.
[29] M. T. Goodrich, Y. Matias, and U. Vishkin, *Optimal parallel approximation algorithms for prefix sums and integer sorting*, in Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms, Jan. 1994, pp. 241–250.
[30] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics*, Addison–Wesley, Reading, MA, May 1989.
[31] V. Grolmusz and P. L. Ragde, *Incomparability in parallel computation*, Discrete Appl. Math., 29 (1990), pp. 63–78.
[32] T. Hagerup, *Towards optimal parallel bucket sorting*, Inform. and Comput., 75 (1987), pp. 39–51.
[33] T. Hagerup and T. Radzik, *Every robust CRCW PRAM can efficiently simulate a Priority PRAM*, in 2nd Annual ACM Symposium on Parallel Algorithms and Architectures, 1990, pp. 117–124.
[34] J. JáJá, *Introduction to Parallel Algorithms*, Addison–Wesley, Reading, MA, 1992.
[35] R. M. Karp, M. Luby, and F. Meyer auf der Heide, *Efficient PRAM simulation on a distributed memory machine*, in Proc. 24th Annual ACM Symposium on Theory of Computing, 1992, pp. 318–326.
[36] D. E. Knuth, *Sorting and Searching*, vol. 3, Addison–Wesley, Reading, MA, 1973.
[37] Y. Matias and U. Vishkin, *Converting high probability into nearly-constant time—with applications to parallel hashing*, in Proc. 23rd Annual ACM Symposium on Theory of Computing, May 1991, pp. 307–316.
[38] Y. Matias and U. Vishkin, *On parallel hashing and integer sorting*, J. Algorithms, 12 (1991), pp. 573–606.
[39] K. Mehlhorn, *Data Structures and Algorithms* I: *Sorting and Searching*, Springer-Verlag, Berlin, Heidelberg, 1984.
[40] K. Mehlhorn and A. Tsakalidis, *Data structures*, in Handbook of Theoretical Computer Science, J. van Leeuwen, ed., vol. A, North–Holland, Amsterdam, 1990, pp. 301–341.
[41] Y. Shiloach and U. Vishkin, *An $O(\lg n)$ parallel connectivity algorithm*, J. Algorithms, 3 (1982), pp. 57–67.
[42] A. Siegel, *On universal classes of fast high performance hash functions, their time-space tradeoff, and their applications*, in Proc. 30th IEEE Annual Symposium on Foundation of Computer Science, 1989, pp. 20–25.
[43] L. G. Valiant, *General purpose parallel architectures*, in Handbook of Theoretical Computer Science, J. van Leeuwen, ed., vol. A, Elsevier Science Publishers B.V., Amsterdam, 1990, pp. 944–971.

SIAM J. COMPUT.
Vol. 27, No. 5, pp. 1376–1419, October 1998
© 1998 Society for Industrial and Applied Mathematics
010

A FILTER MODEL FOR CONCURRENT λ-CALCULUS*

MARIANGIOLA DEZANI-CIANCAGLINI†, UGO DE'LIGUORO†, AND ADOLFO PIPERNO‡

Abstract. Type-free lazy λ-calculus is enriched with angelic parallelism and demonic nondeterminism. Call-by-name and call-by-value abstractions are considered and the operational semantics is stated in terms of a must convergence predicate. We introduce a type assignment system with intersection and union types, and we prove that the induced logical semantics is fully abstract.

Key words. λ-calculus, parallelism, nondeterminism, full abstraction, functional programming, concurrency

AMS subject classifications. 03B15, 03B40, 03G10, 68N15, 68Q10, 68Q42, 68Q55

PII. S0097539794275860

1. Introduction.

Powerful computer architectures make parallelism and concurrency feasible. To exploit these features in existing high-level programming languages, while retaining abstraction and logical clarity in writing programs, it is natural to extend those languages by new concepts and constructs. In particular, much work has been done to accommodate parallel and concurrency primitives inside functional programming languages like CML [60] and FACILE [28] (see [29] for further work in the area and for references).

This extension gives rise to the problem of introducing nonfunctional features in the functional framework. To illustrate this, let us consider parallelism first. If the parallel construct is a control primitive which allows the programmer to force the parallel evaluation of two or more arguments to be passed to a function, then the treatment of divergence (and the value passing mechanism) becomes much more complex. For example, a binary function may be undefined if both its arguments are undefined, without being strict in either the first or the second argument. A typical example is Scott's *parallel-or* function (see [62, p. 437]), the binary partial function of booleans that returns true if at least one of its arguments is defined and equal to true, and returns false if both arguments are defined and equal to false.

The *parallel-or* can be further analyzed as an example of parallel composition of compatible sequential functions. Indeed, let

$$Lor \equiv \lambda xy.\ \textbf{if}\ x\ \textbf{then}\ \text{true}\ \textbf{else}\ y\ \textbf{fi}, \qquad Ror \equiv \lambda xy.\ \textbf{if}\ y\ \textbf{then}\ \text{true}\ \textbf{else}\ x\ \textbf{fi}$$

be the left-sequential and the right-sequential *or*, respectively. Then these functions are compatible, since, in the pointwise ordering induced by the flat domain of booleans, they have an upper bound (actually a join) which is the *parallel-or* function itself. On the other hand, if they can be computed in parallel, returning as soon as either the computation of *Lor* or the computation of *Ror* stops, then we have an implementation of the parallel-or function.

* Received by the editors October 19, 1994; accepted for publication (in revised form) July 31, 1996; published electronically May 19, 1998. This work has been partially supported by grants from ESPRIT-BRA 7232 GENTZEN and from CNR-GNASAGA. A preliminary version of this paper appeared in Proceedings of TACS'94, Lecture Notes in Comput. Sci. 789, Springer-Verlag, New York, 1994, pp. 16–35.

http://www.siam.org/journals/sicomp/27-5/27586.html

† Dipartimento di Informatica, Università di Torino, Corso Svizzera 185, 10149 Torino, Italy (dezani@di.unito.it, deligu@di.unito.it).

‡ Dipartimento di Scienze dell'Informazione, Università di Roma "La Sapienza," Via Salaria 113, 00198 Roma, Italy (piperno@dsi.uniroma1.it).

If parallel composition is a binary operator that can be applied to any pair of functions—not necessarily compatible—then the same evaluation mechanism is a nondeterministic device that can be modeled as a multivalued function. An example is McCarthy's *amb* function [47]. This kind of multivalued function has been widely considered in the literature. In the folklore this form of nondeterminism is called angelic nondeterminism (and credited to Hoare) because of its behavior with respect to divergence: a parallel composition is convergent if at least one of its operands converges. In terms of Dijkstra's correctness criteria, this corresponds to partial correctness.

Concurrency has been added to functional languages using CCS- or CSP-like synchronization and communication primitives. In both cases the interaction with the environment introduces a different form of nondeterminism, as unpredictable events may affect the behavior of the system. In particular, nondeterminism comes in when a choice occurs among guarded commands having the same guard (see [38]).

This nondeterminism has been modeled using internal choice operators, which are correctly considered as abstraction or specification tools. Indeed, no programmer may wish to use internal choice to control the evaluation of a program; it has to be thought of instead as a declaration, saying that, whatever the actual alternative will be, the program still satisfies the correctness requirements. Of course, the criterion is that of total correctness, so that, with respect to divergence, an internal choice is divergent as soon as one of its operands diverges. In folkloric terms, this is demonic nondeterminism. A survey about nondeterminism in functional languages can be found in [65].

When facing these theoretical problems a primary point is to choose the abstraction level of the investigation. One may take a very abstract view and consider them as multifunctions, or, equivalently, functions over powerdomains. This study has been pioneered by Plotkin in [58] and pursued by several authors (see [66] and, for a survey, [46]). Here continuity is the only aspect of computation which is retained in the theory, the main point being the treatment of divergence.

An alternative and quite concrete approach is to model functionality, concurrency, and parallelism by syntactical tools. This amounts to design theoretical languages that formalize essentially all aspects of the computation and interaction, so that actual programming languages can be seen as sugared syntax of the former ones. In this case the languages and the related calculi are inspired on one hand by the λ-calculus, both typed and type-free, and on the other hand by the process calculi (CCS, CSP, ACP, etc.). In exploiting the "concrete" approach, there are at least two main streams. Following the first, functions and processes are first-class objects. The resulting calculus can either be seen as a λ-calculus with processes as possible arguments of functions (as in Nielson's TPL [53]) or as a process algebra with a special form of communication, generalizing the β-reduction of the λ-calculus (as in Thomsen's CHOCS [67, 34, 35]). The second stream does not allow processes as arguments of functions: instead, channels (or port names) have a first-class status and can be sent as values (see, e.g., [11]). The most radical step in this direction is to think of processes just as agents that communicate each others' channel names as values. In this way processes are virtually passed by sending the name of a (private) channel to the receiver, thus giving access to the "passed" process: this is Milner's π-calculus [51]. In the latter case, functions and functional application disappear from the calculus syntax, and they are simulated in a rather complex way.

In this paper we advocate a third approach to the problem of the mathematical study of relevant aspects of concurrent functional languages, which, in some sense,

sits in between the abstract denotational method and the concrete, direct description of interaction and communication. In this case one still considers a formal language together with its operational semantics. The latter gives an essential (and effective) description of the evaluation of expressions in the language. The main departure from the concrete approach, however, is the abstraction from communication, concentrating on a syntax which represents different kinds of nondeterminism by means of different operators, whose behavior is axiomatically described by the rules of the operational semantics.

In this perspective the interaction between functionality and nondeterminism has been studied both in the algebraic framework of rewriting [18, 31, 32, 2], where no abstraction operator is present, and in the λ-calculus framework, either typed [9, 10, 64] or type-free [19, 45, 25, 56, 26, 8].

1.1. Angelic nondeterminism. Our study confronts various problems that had their origin in the theory of functional languages and λ-calculus. In [59] Plotkin showed that Scott continuous functions over domains are overabounding to give meaning to the sequential functional language that has been called PCF (a simply typed λ-calculus with arithmetical constants, booleans, if-then-else, and fixed-point operator). To be precise, he considered the following notion of operational equivalence. Two terms, M and N, of the same type are *operationally equivalent* if and only if, for all contexts $C[\,]$ of ground type such that both $C[M]$ and $C[N]$ are well-typed closed terms, either the evaluations of $C[M]$ and $C[N]$ do not terminate (converge), or both terminate and give the same result. It turns out that, if two terms have the same denotation in the standard model (in which ground types are flat cpo's and arrow types are interpreted as spaces of Scott continuous functions) then they are operationally equivalent (adequacy theorem); but the converse (full abstraction) does not hold.

In the same paper Plotkin proved that syntax can be reasonably enriched to get full abstraction, and that this can be achieved by using a suitable kind of parallel operator or combinator. Milner proved in [49] that this is also a necessary condition: any model of PCF is fully abstract if and only if all "finite" objects in the model are definable. Conversely, the standard model becomes fully abstract if we endow the calculus with operators that reinforce its expressive power such that it satisfies the Milner definability requirement.

The same incompleteness phenomenon with respect to standard continuous semantics has been found for the lazy λ-calculus in [6]. This is a type-free calculus, having the same syntax of pure λ-calculus and a reduction relation over closed terms, with just two rules:

$$(\lambda x.M)N \longrightarrow M[N/x] \qquad \text{and} \qquad \frac{M \longrightarrow M'}{MN \longrightarrow M'N}.$$

The full abstraction problem can be reformulated in this setting, even if we do not have the notion of ground type. Indeed, Abramsky and Ong [6] define the set Val of values as the set of abstractions. Then their notion of *may* convergence is: M may converge to V, written $M \Downarrow^{\text{may}} V$, if V is a value and $M \longrightarrow^* V$. In [6] the operational semantics is given by axiomatizing $M \Downarrow^{\text{may}} V$, instead of giving the reduction relation as a primitive notion; of course, this is equivalent.

As a matter of fact, the problem of enriching the calculus so that the standard model is fully abstract can be solved by adding a combinator $\mathbf{P}$ testing convergence in parallel. More precisely, $\mathbf{P}$ satisfies

$$[\exists V,\ M \Downarrow^{\text{may}} V \text{ or } N \Downarrow^{\text{may}} V] \quad \Rightarrow \quad \mathbf{P}MN \Downarrow^{\text{may}} \mathbf{I},$$

where $\mathbf{I} \equiv \lambda x.x$ is the identity combinator. This gives a combinator which tests convergence, i.e., a closed term $\mathbf{C}$ such that, for any term M, $\mathbf{C}M$ reduces to $\mathbf{I}$ if M reduces to a value and diverges otherwise; just take $\mathbf{C} \stackrel{\text{Def}}{=} \lambda x.\mathbf{P}xx$.

In [19] a further step is made by Boudol. The combinator $\mathbf{P}$ is split into its two components, namely, parallelism and convergence test. The parallelism implicit in $\mathbf{P}$ is made explicit by adding a binary operator $\|$ such that

$$M\|N \Downarrow^{\text{may}} \;\Leftrightarrow\; M \Downarrow^{\text{may}} \text{ or } N \Downarrow^{\text{may}},$$

where $M \Downarrow^{\text{may}}$ abbreviates $\exists V,\ M \Downarrow^{\text{may}} V$. To have this, with the above definition of convergence, the following rules suffice:

$$\frac{M \longrightarrow M'}{M\|N \longrightarrow M'\|N} \quad \text{and} \quad \frac{N \longrightarrow N'}{M\|N \longrightarrow M\|N'}.$$

As the intended meaning of a parallel composition is a function, Boudol adds the following rule:

$$(M\|N)L \longrightarrow (ML)\|(NL).$$

The internal convergence test is achieved using, besides standard call-by-name abstraction, call-by-value abstraction, originally considered by Landin [41] and Plotkin [57]. To see how this works, let us extend the set Val of values inductively so that it includes all terms of the shape $V\|N$ or $M\|V$, where V is a value. We use two sorts of variables to distinguish between call-by-value and call-by-name abstraction, namely, $v, w, \ldots$ for call-by-value variables and $x, y, \ldots$ for call-by-name variables. Then we add to the lazy λ-calculus and to the rules for $\|$, the following rules:

$$(\lambda v.M)V \longrightarrow M[V/v] \text{ if } V \in \mathsf{Val}, \qquad \frac{N \longrightarrow N'}{(\lambda v.M)N \longrightarrow (\lambda v.M)N'} \text{ if } N \notin \mathsf{Val}.$$

Now $\mathbf{P}$ becomes definable by $\lambda xy.\ (\lambda v.\mathbf{I})(x\|y)$.

We observe that the combination of parallelism (angelic nondeterminism) and call-by-value is much more powerful than the use of combinators directly defining a parallel convergence test. First, the notion of being a value is no more equivalent to that of being irreducible. Moreover, as remarked in the early paragraphs of this introduction, $M\|N$ has to be interpreted as a multivalued function, since M and N are not necessarily interpreted by compatible functions. So the model of [19] is a solution of the domain equation $D = \mathcal{P}^\flat([D \to D]_\perp)$, where $[D \to D]_\perp$ is the lifted space of continuous functions, and $\mathcal{P}^\flat$ is the lower powerdomain functor (also called Hoare's powerdomain; see [66] for a definition). Since Boudol works in the category of prime algebraic lattices, he has this solution for free. In fact, in that category $D \simeq \mathcal{P}^\flat(\mathsf{KP}(D))$, where $\mathsf{KP}(D)$ is the set of compact coprime elements of D. Let us recall that a *complete lattice* is a partial order $(D, \sqsubseteq)$ such that each subset X of D has a least upperbound $\bigsqcup X$. An element d of a complete lattice is *compact* if $d \sqsubseteq \bigsqcup X$ implies $d \sqsubseteq \bigsqcup Y$ for some finite subset Y of X. An element $d \in D$ is *coprime* if and only if $d \sqsubseteq x \sqcup y$ implies $d \sqsubseteq x$ or $d \sqsubseteq y$. A complete lattice is *prime algebraic* if any element is the join of the compact coprime elements it dominates. See also the discussion at the beginning of section 4.

1.2. Demonic nondeterminism. Serious problems arise when we consider the full language, modeling also the demonic nondeterminism (see [55, 56]), which is the central issue of the present paper. Suppose that an internal choice operator $+$ is added, with the obvious reduction rules

$$M + N \longrightarrow M \quad \text{and} \quad M + N \longrightarrow N.$$

Then, following ideas explained above (see also [45]), we expect a convergence predicate $\Downarrow$ such that

$$M + N \Downarrow \quad \Leftrightarrow \quad M \Downarrow \quad \text{and} \quad N \Downarrow .$$

But this is not true with the present definition of $\Downarrow^{\text{may}}$.

The convergence predicate considered above (and in [19]) is a *may* convergence predicate, to be related to may testing equivalence if convergence is the only observable property (see [3, 33, 6]). A solution would be to consider a *must* convergence predicate as in [45] (see also [35]). An informal definition is the following: $M \Downarrow^{\text{must}}$ if and only if there is an n such that every reduction out of M reaches a value within a number of steps bounded by n. Otherwise we write $M \Uparrow^{\text{must}}$.

Of course, if we have to avoid the collapse of $\|$ and $+$ with respect to the predicate $\Downarrow^{\text{must}}$, something has to be changed in the operational semantics of $\|$. In fact, with the old definition of $\|$-reduction rules, if we put, for example, $\Delta \equiv \lambda x.xx$ and we take the typical divergent combinator $\Omega \equiv \Delta\Delta$, then we have that $(\mathbf{II})\|\Omega \Uparrow^{\text{must}}$. The problem is that nothing prevents the reduction of a parallel composition from being unfair: there exists a reduction out of $(\mathbf{II})\|\Omega$ that contracts Ω infinitely many times and never reaches the value $\mathbf{I}\|\Omega$. Really, we want to identify $\mathbf{I}\|\Omega$ with $\mathbf{I}$, since $\|$ is intended to take the best of its arguments; notice that the above-mentioned terms are not equivalent in a standard must semantics (see [24]), when the parallel operator is asynchronous.

There are many possibilities for changing the reduction rules for $\|$ in such a way that we cannot reduce infinitely many times on one side of a parallel composition, when the other one is reducible. We take the simplest way to get this kind of *fair* reduction and we introduce the rules

$$\frac{M \longrightarrow M' \quad N \longrightarrow N'}{M\|N \longrightarrow M'\|N'}, \qquad \frac{M \longrightarrow M' \quad N \not\longrightarrow}{M\|N \longrightarrow M'\|N, \quad N\|M \longrightarrow N\|M'}$$

as our actual choice (see [25]), where $N \not\longrightarrow$ means that N is irreducible.

This implies that, as in [19], a value of the shape $V\|M$ is not necessarily a normal form, as M can be reduced. This fact, together with the presence of the choice operator, makes the β-rule for call-by-value sensible to the relative speed of parallel evaluations of its arguments.

To illustrate this, let us consider the context $C[\ ] \equiv (\lambda v.vv)[\ ]\Omega\mathbf{I}$ and the values $V \equiv \mathbf{I}\|(\mathbf{K} + \mathbf{O})$, $V' \equiv \mathbf{I}\|\mathbf{K}$, and $V'' \equiv \mathbf{I}\|\mathbf{O}$, where $\mathbf{K} \equiv \lambda xy.x$ and $\mathbf{O} \equiv \lambda xy.y$. Then $V \longrightarrow V'$ and $V \longrightarrow V''$. Now (writing $\xrightarrow{n}$ for $n > 1$ reduction steps)

$$\begin{array}{lcl}
C[V'] & \longrightarrow & (\mathbf{I}\|\mathbf{K})(\mathbf{I}\|\mathbf{K})\Omega\mathbf{I} \\
& \xrightarrow{3} & (\mathbf{I}(\mathbf{I}\|\mathbf{K})\Omega\mathbf{I})\|(\mathbf{K}(\mathbf{I}\|\mathbf{K})\Omega\mathbf{I}) \\
& \longrightarrow & ((\mathbf{I}\|\mathbf{K})\Omega\mathbf{I})\|((\lambda y.(\mathbf{I}\|\mathbf{K}))\Omega\mathbf{I}) \\
& \longrightarrow & ((\mathbf{I}\Omega\|\mathbf{K}\Omega)\mathbf{I})\|((\mathbf{I}\|\mathbf{K})\mathbf{I}) \\
& \longrightarrow & (\mathbf{I}\Omega\mathbf{I})\|(\mathbf{K}\Omega\mathbf{I})\|(\mathbf{II})\|(\mathbf{KI}) \\
& \longrightarrow & (\Omega\mathbf{I})\|((\lambda y.\Omega)\mathbf{I})\|\mathbf{I}\|(\lambda y.\mathbf{I}) \\
& \longrightarrow & (\Omega\mathbf{I})\|\Omega\|\mathbf{I}\|(\lambda y.\mathbf{I}),
\end{array}$$

which is a value, and it is not hard to see that this is the only reduction out of $C[V']$ according to the rules defined in Definition 2.2. Similarly,

$$\begin{array}{lll} C[V''] & \longrightarrow & (\mathbf{I}\|\mathbf{O})(\mathbf{I}\|\mathbf{O})\Omega\mathbf{I} \\ & \longrightarrow^* & (\Omega\mathbf{I})\|\mathbf{I}\|(\Omega\mathbf{I}), \end{array}$$

and again this is the only reduction out of $C[V'']$. But now consider the following reduction of $C[V]$:

$$\begin{array}{lll} C[V] & \longrightarrow & (\mathbf{I}\|(\mathbf{K}+\mathbf{O}))(\mathbf{I}\|(\mathbf{K}+\mathbf{O}))\Omega\mathbf{I} \\ & \xrightarrow{3} & (\mathbf{I}(\mathbf{I}\|(\mathbf{K}+\mathbf{O}))\Omega\mathbf{I})\|((\mathbf{K}+\mathbf{O})(\mathbf{I}\|(\mathbf{K}+\mathbf{O}))\Omega\mathbf{I}) \\ & \longrightarrow & ((\mathbf{I}\|(\mathbf{K}+\mathbf{O}))\Omega\mathbf{I})\|(\mathbf{O}(\mathbf{I}\|(\mathbf{K}+\mathbf{O}))\Omega\mathbf{I}) \qquad \text{choosing } \mathbf{O} \ldots \\ & \xrightarrow{2} & (\mathbf{I}\Omega\mathbf{I})\|((\mathbf{K}+\mathbf{O})\Omega\mathbf{I})\|(\Omega\mathbf{I}) \\ & \longrightarrow & (\Omega\mathbf{I})\|(\mathbf{K}\Omega\mathbf{I})\|(\Omega\mathbf{I}) \qquad \ldots \text{choosing } \mathbf{K} \\ & \longrightarrow & (\Omega\mathbf{I})\|\Omega\|(\Omega\mathbf{I}), \end{array}$$

and from $(\Omega\mathbf{I})\|\Omega\|(\Omega\mathbf{I})$ we will never reach a value.

This example also shows that there are values V_0, V_1, and V_2 such that $V_0\|(V_1+V_2)$ and $(V_0\|V_1)+(V_0\|V_2)$ would have different behaviors in some context, although this would be unexpected under any reasonable operational semantics. $(\lambda v.vv)(V_0\|(V_1 + V_2))$ can indeed reduce to $(V_0\|(V_1 + V_2))(V_0\|(V_1 + V_2))$, while $(\lambda v.vv)((V_0\|V_1) + (V_0\|V_2))$ can reduce either to $(V_0\|V_1)(V_0\|V_1)$ or to $(V_0\|V_2)(V_0\|V_2)$, but never to $(V_0\|(V_1 + V_2))(V_0\|(V_1 + V_2))$. Note that in the present context call-by-name and call-by-value implement run-time-choice and call-time-choice, respectively (see [45]).

The problem of correcting the β-contraction rule for call-by-value is that, given a value V, we cannot decide whether it has been computed enough to perform the reduction step $(\lambda v.M)V \longrightarrow M[V/v]$, or if it is necessary to reduce V further, before contracting the outermost β-redex. We cannot reduce V as long as possible, since this could not terminate. In the meantime, $M[V/v]$ can diverge while $M[V'/v]$ can converge for all V' which are reducts of V, as shown by the previous example. On the other hand, any effective description of the operational semantics calls for a definition of a recursive one-step reduction relation.

Now the solution we propose is to distinguish two cases: if V is an irreducible value (namely a λ-abstraction or the parallel composition of irreducible values), then the standard call-by-value β-contraction rule applies. If, instead, V can be reduced further, to compute $(\lambda v.M)V$ we want to "take the best" between the terms $M[V'/v]$, for all V' such that $V \longrightarrow^* V'$. We realize this by evaluating in parallel $M[V/v]$ and $(\lambda v.M)V'$ for all V' such that $V \longrightarrow V'$. Using the operator $\|$, this can be formalized in our calculus as follows:

$$\frac{V \not\longrightarrow \qquad V \in \mathsf{Val}}{(\lambda v.M)V \longrightarrow M[V/v]}, \qquad \frac{V \longrightarrow V' \qquad V \in \mathsf{Val}}{(\lambda v.M)V \longrightarrow M[V/v]\|(\lambda v.M)V'}.$$

In other words, the solution we propose is to distinguish between *total* and *partial* values. A total value is an irreducible value, while a partial value is of the form $M\|N$ in which either M or N is not a total value. So we split the call-by-value β-contraction in two rules.

To conclude this part of our discussion, let us emphasize the effectiveness of the evaluation mechanism as a distinguishing feature of our calculus. As is clear from the previous exposition, the papers closest to the present one are [19] and [55].[1] While our

[1] Following Ong's paper we named our calculus *concurrent λ-calculus*.

treatment improves on the former because of the presence in the same calculus of both angelic and demonic nondeterminism, it improves on the latter since the operational semantics on which we base our theory is effective. Indeed, the reduction relation is (as usual) presented by means of a formal system in the sense of Post, and the convergence predicate is (up to coding) recursively enumerable. This is mandatory when one expects to capture the intentional aspects of evaluation and justifies our reduction relation as it will be defined in the technical development of the paper.

1.3. Intersection and union types. The complex operational semantics of the concurrent λ-calculus asks for an abstract treatment not involving direct reasoning on possible reducts of a given term. The approach taken in this paper is to use a type assignment system that sufficiently expresses the operational equivalence of terms. We expect that M and N have exactly the same types when they have the same behavior in any context: this is a fully abstract "logical" semantics in the sense of [63], [16], and [5].

To this aim, we use a system with intersection and union types, dually reflecting the disjunctive and conjunctive operational semantics of $\|$ and $+$. Types are viewed as properties of terms concerning their behavior with respect to the convergence predicate and type inclusion as the logical implication. The system has a universal type ω, the property which always holds; therefore, any type will be less than ω. As usual with type assignment systems for polymorphic λ-calculi, the arrow type expresses functionality: M has type $\sigma \to \tau$ if, for all N having type σ, MN has type τ. With respect to the order, the arrow is covariant in the second argument and contravariant in the first argument. Finally, $\sigma \wedge \tau$ and $\sigma \vee \tau$ have a conjunctive and disjunctive meaning, respectively.

The lazy semantics distinguishes between functions (even the everywhere undefined function) and the undefined object, representing divergence. This means that the interpretation of the term $\lambda x.\Omega$ is better than the interpretation of Ω. These terms, instead, are equated in the theory of solvability of the classical λ-calculus [15]. On the side of types, this distinction is modeled by making the inclusion $\omega \to \omega \leq \omega$ proper. As a matter of fact, among the axioms in [16] concerning the arrow, we save $\sigma \to \omega \leq \omega \to \omega$, which makes $\omega \to \omega$ the type of all functions, but we reject $\omega \leq \omega \to \omega$, which would equate the interpretations of the terms Ω and $\lambda x.\Omega$ (see Corollary 5.6(ii)).

We now turn to the typing rules for nondeterministic and parallel operators. We know that the term $M + N$ can be reduced to both M and N, so that to ensure correctness we have to prove that both M and N have the same type σ before we can conclude that $M + N$ has type σ (this is also the choice of [1]). Extending the disjunctive semantics of the parallel composition from convergence to arbitrary properties, it follows that one is entitled to type $M\|N$ with σ as soon as M or N (or both) can be typed with σ (see [19] for further explanations). This suggests the following typing rules:

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M + N : \sigma}, \qquad \frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M\|N : \sigma}, \qquad \frac{\Gamma \vdash N : \sigma}{\Gamma \vdash M\|N : \sigma}.$$

The inclusion relation $\leq$ among types makes $\wedge$ into the meet and $\vee$ into the join, and we have both a subtyping and an intersection rule, namely,

$$\frac{\Gamma \vdash M : \sigma \quad \sigma \leq \tau}{\Gamma \vdash M : \tau}, \qquad \frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash M : \tau}{\Gamma \vdash M : \sigma \wedge \tau}.$$

Therefore, the rules for $+$ and $\|$ above are equivalent to

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash M + N : \sigma \vee \tau}, \qquad \frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash M \| N : \sigma \wedge \tau},$$

which is our actual choice.

If M has type $\sigma \vee \tau$ then it can be that M evaluates to both some P and some Q such that P has type σ and Q has type τ, but neither P has type τ nor Q has type σ. In this case, M has an essentially disjunctive type, which is possible even if M is a partial value. But all is determined in the case of total values. So we expect the system to have the "disjunction property" for total values: if a total value has the type $\sigma \vee \tau$, then either σ or τ can be assigned to it (hence to all its reducts).

Consequently, we distinguish between call-by-name and call-by-value abstraction, making a substantial use of disjunction. This intuitively explains why the rule

$$\frac{\Gamma \vdash \lambda v.M : (\sigma \to \rho) \wedge (\tau \to \rho)}{\Gamma \vdash \lambda v.M : \sigma \vee \tau \to \rho}$$

is correct for call-by-value but not for call-by-name abstraction. Observe that this means that call-by-value abstraction yields a co-additive function (namely, meet preserving), which is the expected semantics of call-by-value in our setting.

As an example, if $M \equiv (x\mathbf{I}\Omega)\|(x\Omega\mathbf{I})$, $\rho \equiv \omega \to \omega$, $\sigma \equiv \rho \to \omega \to \rho$, $\tau \equiv \omega \to \rho \to \rho$, then we have that $\vdash \lambda x.M : (\sigma \to \rho) \wedge (\tau \to \rho)$. Moreover, $\vdash \mathbf{K} : \sigma$ and $\vdash \mathbf{O} : \tau$, so that allowing the rule above for call-by-name abstraction, one could deduce $(\lambda x.M)(\mathbf{K} + \mathbf{O}) : \rho$, using the rules for $+$ introduction and $\to$ elimination, too. But this would destroy the subject-reduction property, since $(\lambda x.M)(\mathbf{K} + \mathbf{O})$ reduces to $\Omega\|\Omega$, for which only types equivalent to ω can be deduced (see Corollary 5.6(ii)).

The type assignment system implicitly suggests a notion of interpretation in which each term can be seen as denoting the set of types it can be assigned. Then one can think of extending the notion of filter models such that they encompass the present calculus and union types.

Filter models were introduced in [16] for the classical λ-calculus and they were based on the intersection-type discipline. In that case, however, discovering that filters of types do actually form a structure (a λ-model) was based on the pre-existing and independent definition of this kind of mathematical structure (see [37, 48]). Here the problem is the opposite: given the logical interpretation induced by our system, we look for a reasonable definition of what is a model of our calculus.

In the extended view of Curry types (see [16, 21]), type theories are an instance of information systems (see [63, 23]). Taking filters of types, we have a domain that, seen topologically, is the Stone space generated by the theory of type inclusion (see [40, 5]). In the present case the domain that is determined by the type theory we consider is isomorphic to the initial solution of the domain equation $D = \mathcal{P}^\sharp([D \to D]_\perp)$ in the category of continuous lattices, where $\mathcal{P}^\sharp$ is the upper powerdomain functor (also called Smyth's powerdomain; see [66]). This is sound with respect to the operational semantics since this powerdomain constructor is needed to model demonic nondeterminism, as angelic nondeterminism is built in, by the fact that we work with prime algebraic lattices (as remarked at the end of subsection 1.1). This domain equation, and their relations to Abramsky and Boudol equations [5, 19], will be discussed further at the beginning of section 4.

We do not carry out the details of the isomorphism between the filter model and the initial solution of the given domain equation, for which we refer to [7]. Instead, we

analyze compositionally the interpretation of terms defined by $[\![M]\!] = \{\sigma \mid \vdash M : \sigma\}$ (where M is closed), and devise a category of objects that embodies the minimum needed structure to interpret the calculus. We then get a notion of environment model for the present calculus, in the sense of [37]. The filter model induced by our type assignment turns out to fit into this notion, a fact that will be used to prove completeness of type inference.

Our study culminates in the full abstraction theorem, which we will prove by means of characteristic terms extending [19].

1.4. Summary. In section 2 we formally define the concurrent λ-calculus and its reduction rules. We consider the reduction trees of terms to introduce convergence. Moreover, we consider another reduction relation, whose main feature is to characterize convergent terms as those which reduce to a sum of values.

Section 3 deals with types and the type assignment system. The choice of the preorder on types, which will determine the topological structure of the filter model, is crucial. The type assignment system turns out to enjoy structural properties which allow us to prove preservation of type under subject reduction. The main result of this section is that all convergent terms can be typed by $\omega \to \omega$.

Section 4 presents the filter model as the initial solution of a suitable domain equation. Then we introduce the notion of environment model for concurrent λ-calculus and we prove that the filter model is in fact an environment model. This allows us to have the completeness of type assignment.

Finally, we prove in section 5 the full abstraction of the filter model. First we define for each type a test term and a characteristic term. The application of the test term to an argument M converges only if M has the corresponding type. By means of a realizability interpretation of types we show that all terms typed by $\omega \to \omega$ converge. This, together with the main result of section 3, implies that $\omega \to \omega$ completely characterizes convergence. Then the full abstraction of the model follows easily.

2. The calculus and its operational semantics. We extend the syntax of pure λ-calculus with a non-deterministic choice operator $+$ and a parallel operator $\|$. We use two sorts of variables, namely, the set Vn of call-by-name variables, ranged over by x, y, z, and the set Vv of call-by-value variables, ranged over by v, w. The symbol χ will range over the set $\mathsf{Vn} \cup \mathsf{Vv}$. The terms of the concurrent λ-calculus are defined by the following grammar:

$$M ::= x \mid v \mid (\lambda x.M) \mid (\lambda v.M) \mid (MM) \mid (M + M) \mid (M \| M).$$

We call $\Lambda_{+\|}$ the set of terms. For any $M \in \Lambda_{+\|}$, $FV(M)$ denotes the set of free variables of M and $\Lambda^0_{+\|}$ is the set of terms M such that $FV(M) = \emptyset$. Moreover, we shall refer to the following set:

$$\mathsf{Par} = \{(M \| N) \mid M, N \in \Lambda_{+\|}\}.$$

Notation. We use $\equiv$ for syntactical equality up to renaming of bound variables. As usual for pure λ-calculus, we assume that application associates to the left, and we write, e.g., MNP instead of $((MN)P)$. If $\vec{L} \equiv L_1 \dots L_n$ is any (possibly empty) vector of terms, then $M\vec{L} \equiv ML_1 \dots L_n$. The expression $\lambda\chi_1 \dots \chi_n.M$ is short for $(\lambda\chi_1.(\dots(\lambda\chi_n.M)\dots))$.

We will abbreviate some λ-terms as follows:

$$\mathbf{I} \stackrel{\text{Def}}{=} \lambda x.x, \quad \mathbf{K} \stackrel{\text{Def}}{=} \lambda xy.x, \quad \mathbf{O} \stackrel{\text{Def}}{=} \lambda xy.y,$$
$$\Delta \stackrel{\text{Def}}{=} \lambda x.xx, \quad \Omega \stackrel{\text{Def}}{=} \Delta\Delta, \quad \mathbf{Y} \stackrel{\text{Def}}{=} \lambda y.(\lambda x.y(xx))(\lambda x.y(xx)).$$

Application and abstraction have precedence over + and ‖; e.g., $MN + P$ stands for $((MN) + P)$, and $\lambda x.M + N$, for $((\lambda x.M) + N)$. The operator ‖ takes precedence over +; for example, $M\|P + Q$ is short for $((M\|P) + Q)$. External parentheses are always omitted.

The operators + and ‖ will be written up to associativity. We shall also make use of the following abbreviation:

$$\sum_{i=1}^{n} M_i \equiv M_1 + \cdots + M_n.$$

Moreover, if $\mathcal{M} = \{M_1, \ldots, M_n\}$ is any finite multiset of terms then

$$\sum \mathcal{M} \equiv \sum_{i=1}^{n} M_i.$$

Observe that, $\mathcal{M}$ being a multiset, it can be the case that $M_i \equiv M_j$ for different i and j.

As discussed in the introduction, we need to distinguish between partial and total values; the main difference concerns the parallel operator. In fact, we require both M and N to be total values to ensure that $M\|N$ is a total value, while in general it suffices that either M or N be a value for $M\|N$ to be a value. As is clear from the next definition, a value is either a total or a partial value.

DEFINITION 2.1. *We define the set* Val *of* values *according to the grammar*

$$V ::= v \mid \lambda x.M \mid \lambda v.M \mid V\|M \mid M\|V$$

and the set TVal *of* total values *as the subset of* Val*:*

$$W ::= v \mid \lambda x.M \mid \lambda v.M \mid W\|W\ .$$

A value V is partial *if and only if $V \notin$ TVal.*

We now introduce a reduction relation which is intended to formalize the expected behavior of a machine which evaluates parallel compositions in a synchronous way until a value is produced. Partial values can be further evaluated, and this is essential in the case of an application of a call-by-value abstraction. Therefore, in some cases an asynchronous evaluation of parallel composition is permitted.

It follows that the convergence predicate will not be any more coincident with the property of being (strongly) normalizable (see [19] for a similar proposal, even if in a may perspective) with respect to the given reduction relation. Observe that in the lazy λ-calculus of [6], as well as in the present calculus, $\lambda\chi.M$ is a normal form, no matter whether M is reducible or not.

DEFINITION 2.2.

(i) *The reduction relation $\longrightarrow$ is the least binary relation over $\Lambda^0_{+\|}$ such that*

$$(\beta)\quad (\lambda x.M)N \longrightarrow M[N/x], \qquad (\beta_v)\quad \frac{W \in \mathsf{TVal}}{\lambda v.M)W \longrightarrow M[W/v]},$$

$$(\beta_v\|)\quad \frac{V \longrightarrow V' \qquad V \in \mathsf{Val}}{(\lambda v.M)V \longrightarrow M[V/v]\|(\lambda v.M)V'}, \qquad (\mu_v)\quad \frac{N \longrightarrow N' \quad N \notin \mathsf{Val}}{(\lambda v.M)N \longrightarrow (\lambda v.M)N'},$$

$$(\nu)\quad \frac{M \longrightarrow M' \qquad M \notin \mathsf{Val} \bigcup \mathsf{Par}}{MN \longrightarrow M'N}, \qquad (\|_{app})\quad (M\|N)L \longrightarrow ML\|NL,$$

$$(\|_s)\quad \frac{M \longrightarrow M' \quad N \longrightarrow N'}{M\|N \longrightarrow M'\|N'}, \qquad (\|_a)\quad \frac{M \longrightarrow M' \quad W \in \mathsf{TVal}}{M\|W \longrightarrow M'\|W,\ W\|M \longrightarrow W\|M'},$$

$$(+)\quad M + N \longrightarrow M, \quad M + N \longrightarrow N.$$

(ii) *We denote by* $\longrightarrow^*$ *the reflexive and transitive closure of* $\longrightarrow$.

LEMMA 2.3.

(i) $W \in \Lambda^0_{+\|}$ *is irreducible with respect to* $\longrightarrow$ *if and only if* $W \in \mathsf{TVal}$;

(ii) *If* $V \in \Lambda^0_{+\|} \cap \mathsf{Val}$, *then either* $V \in \mathsf{TVal}$ *or* $V \longrightarrow V'$ *for some* $V' \in \mathsf{Val}$;

(iii) *If* $W, W_1, \ldots, W_n \in \mathsf{TVal}, N_1, \ldots, N_m \in \Lambda_{+\|}$, *then*

$$W[N_1/x_1, \ldots, N_m/x_m, W_1/v_1, \ldots, W_n/v_n] \in \mathsf{TVal}.$$

Proof. The proof is easy by the definitions. □

It is useful to consider reduction trees of terms and their bars.

DEFINITION 2.4. *Let* $M \in \Lambda^0_{+\|}$.

(i) *tree*(M) *is the (unordered) reduction tree of* M;

(ii) *A* bar *of tree*(M) *is a subset of the nodes of tree*(M) *such that each maximal path intersects the bar at exactly one node*;

(iii) *bar*(M) *is the set of bars of tree*(M);

(iv) *For* $b \in bar(M)$ *the* height *of* b *(notation: height*(b)*) is the maximum of the heights of its nodes.*

Inspecting the reduction rules, we see that $tree(M)$ is a finitely branching tree for all $M \in \Lambda^0_{+\|}$. This implies by König's lemma that if we cut $tree(M)$ at a fixed height we obtain a finite tree. Since all nodes belonging to a bar b are in the subtree of $tree(M)$ obtained by cutting $tree(M)$ at $height(b)$, we have that $b \in bar(M)$ is always a finite set of nodes (see also [17]). This does not contradict the fact that a term may have infinite reduction paths. For example, let us consider the infinite reduction tree of $\mathbf{Y}M$, where $M \equiv \lambda x.(\mathbf{I} + x)$, which is shown in Figure 1. Admittedly, the set of nodes in $tree(\mathbf{Y}M)$ which are labeled by $\mathbf{I}$ is infinite, but it is not a bar. Indeed, the infinite path in this tree does not have any node in such a set, and every $b \in bar(\mathbf{Y}M)$ must contain exactly one node of this path. Whichever node we choose on the infinite path, we will exclude all nodes with greater height, so that b turns out to be finite.

A bar is always relative to a tree and cannot be identified with the set of the labels of its nodes. For example, $tree(M + \mathbf{I}M)$ has the shape shown in Figure 2. Now the indicated set of nodes b is a bar whose set of labels is the singleton $\{M\}$. But the set containing a single node labeled by M is not a bar of this tree. Moreover, the height of the bar b is 2, but if b would be identified with $\{M\}$, then $height(b)$ would be ambiguously 1 or 2.

However, if $b \in bar(M)$, then two subtrees rooted in two nodes of b are equal if and only if their labels are the same. Hence we abuse notation and write $b = \{M_1, \ldots, M_n\}$ (if $M_1, \ldots, M_n$ is the multiset of labels of nodes belonging to b). The abbreviations $M \in b$ and $b \subseteq \mathsf{Val}$ will have the obvious meanings.

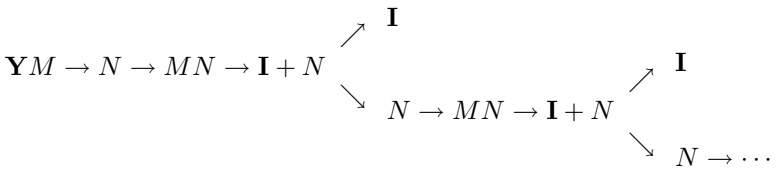


FIG. 1. *Reduction tree of* $\mathbf{Y}M$, *where* $N \equiv (\lambda x.M(xx))(\lambda x.M(xx))$.

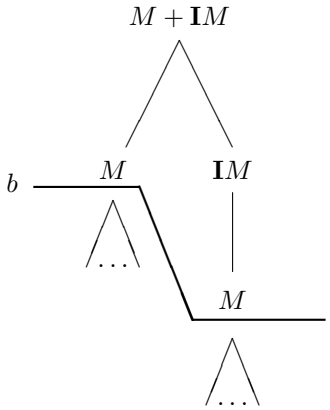


FIG. 2. *Reduction tree of* $M + \mathbf{I}M$.

We now define the convergence predicate. A term is convergent if and only if all reduction paths will eventually reach a value. In other words, a term M converges if and only if there is a bar in $tree(M)$ which is a subset of Val. To formalize this, it is useful to introduce the bar $\mathcal{R}(M,k)$ whose labels are those terms which can be reached starting from a term M by performing k steps of reduction.

DEFINITION 2.5. *Let* $M \in \Lambda^0_{+\|}$; *then*

(i) $\mathcal{R}(M,k) \in bar(M)$ *is the cut of* $tree(M)$ *at height* k; *namely, it is the unique bar such that*
  (a) $height(\mathcal{R}(M,k)) \leq k$;
  (b) $\forall M' \in \mathcal{R}(M,k),\ height(M') < k \ \Rightarrow\ M' \in \mathsf{TVal}$.

(ii) $M\Downarrow_k \Leftrightarrow\ \mathcal{R}(M,k) \subseteq \mathsf{Val}$.

(iii) $M\Downarrow \Leftrightarrow\ \exists k,\ M\Downarrow_k$.

Note that $(M+N)\Downarrow$ if and only if both $M\Downarrow$ and $N\Downarrow$. On the other hand, $(M\|N)\Downarrow$ if and only if either $M\Downarrow$ or $N\Downarrow$ (or both). So $\Downarrow$ coincides with $\Downarrow^{\text{must}}$ as informally defined in section 1.2. In general, if for some $b \in bar(M)$ we have $M'\Downarrow$ for all $M' \in b$, then $M\Downarrow$. The reverse is obviously true.

We depart from the standard way of defining must semantics using infinite paths (see [24]). This gives us a different theory of terms; for example, we equate $\mathbf{I}$ and $\mathbf{I}\|\Omega$.

To study the operational semantics of our calculus it is useful to introduce a binary relation $\rhd$ whose main features are

- to satisfy the Church–Rosser property,

- to simulate the choices performed by rule (+) without losing information about the discarded parts,
- to characterize the convergent terms as those which reduce to a sum of values.

Moreover, we will consider the equivalence relation $\bowtie$ generated by $\rhd$.

DEFINITION 2.6.

(i) *Define* $\rhd$ *as the least binary relation over* $\Lambda^0_{+\|}$ *such that*
$(\beta)'\ (\lambda x.M)N \rhd M[N/x]$,
$(\beta_v)'\ (\lambda v.M)W \rhd M[W/v]$, *if* $W \in \mathsf{TVal}$,
$(\beta_v\|)'\ (\lambda v.M)V \rhd \sum_{i=1}^n (M[V/v]\|(\lambda v.M)V_i)$ *if* $V \in \mathsf{Val}-\mathsf{TVal}$ *and* $\mathcal{R}(V,1) = \{V_1,\dots,V_n\}$,
$(\mu_v)'\ (\lambda v.M)N \rhd \sum_{i=1}^n (\lambda v.M)N_i$ *if* $N \notin \mathsf{Val}$ *and* $\mathcal{R}(N,1) = \{N_1,\dots,N_n\}$,
$(+_{app})'\ (M+N)L \rhd ML + NL$,
$(\|_{app})'\ (M\|N)L \rhd ML\|NL$,
$(\|+)'\ (M+N)\|L \rhd M\|L + N\|L$,
$(\nu)'\ M \rhd M' \Rightarrow MN \rhd M'N$,
$(+)'\ M \rhd M' \Rightarrow M+N \rhd M'+N$,
$(\|)'\ M \rhd M' \Rightarrow M\|N \rhd M'\|N$,
$(+_c)'\ M+N \rhd N+M$,
$(\|_c)'\ M\|N \rhd N\|M$,
$(+_{ass})'\ (M+N)+L \rhd M+(N+L)$ *and* $M+(N+L) \rhd (M+N)+L$,
$(\|_{ass})'\ (M\|N)\|L \rhd M\|(N\|L)$ *and* $M\|(N\|L) \rhd (M\|N)\|L$.

(ii) $\rhd^*$ *is the reflexive and transitive closure of* $\rhd$.

(iii) $\bowtie$ *is the symmetric closure of* $\rhd^*$, *up to associativity and commutativity of* $+$ *and* $\|$.

PROPOSITION 2.7. *The relation* $\rhd$ *is Church–Rosser*; *namely,*

$$\forall M, M_1, M_2 \in \Lambda^0_{+\|},\ M \rhd^* M_1\ \&\ M \rhd^* M_2 \Rightarrow \exists M_3,\ M_1 \rhd^* M_3\ \&\ M_2 \rhd^* M_3.$$

Proof. The proof is a variant of the Tait–Martin-Löf proof for classical λ-calculus (see [15]). We define the following relation on closed terms:

- $M \rightsquigarrow M$;
- if $M \rhd M'$ by any clause among $(\beta)'$, $(\beta_v)'$, $(\beta_v\|)'$, and $(\mu_v)'$ then $M \rightsquigarrow M'$;
- if $M \rightsquigarrow M'$, $N \rightsquigarrow N'$, and $L \rightsquigarrow L'$ then
  - —$(M\|N)L \rightsquigarrow M'L\|N'L$,
  - —$(M+N)L \rightsquigarrow M'L + N'L$,
  - —$(M+N)\|L \rightsquigarrow M'\|L' + N'\|L'$,
  - —$MN \rightsquigarrow M'N$,
  - —$M+N \rightsquigarrow M'+N'$,
  - —$M\|N \rightsquigarrow M'\|N'$,
  - —$M+N \rightsquigarrow N'+M'$,
  - —$M\|N \rightsquigarrow N'\|M'$,
  - —$(M+N)+L \rightsquigarrow M'+(N'+L')$ and $M'+(N'+L') \rightsquigarrow (M'+N')+L'$,
  - —$(M\|N)\|L \rightsquigarrow M'\|(N'\|L')$ and $M'\|(N'\|L') \rightsquigarrow (M'\|N')\|L'$.

By induction on the definition of $\rightsquigarrow$ it is routine to check that it satisfies the diamond property, namely,

$$\forall M, M_1, M_2 \in \Lambda^0_{+\|},\ M \rightsquigarrow M_1\ \&\ M \rightsquigarrow M_2 \Rightarrow \exists M_3,\ M_1 \rightsquigarrow M_3\ \&\ M_2 \rightsquigarrow M_3;$$

hence it is Church–Rosser. Now it is easy to see that $\rightsquigarrow^* = \rhd^*$, from which the thesis follows. □

The relation $\bowtie$ is weaker than the congruence generated by $\rhd$ but stronger than its reflexive, symmetric, and transitive closure. For example, $\mathbf{II} \bowtie \mathbf{I}$, but $(\lambda v.v)(\mathbf{II}\|\mathbf{I}) \not\bowtie (\lambda v.v)(\mathbf{I}\|\mathbf{I})$.

LEMMA 2.8.

(i) *If $M \bowtie N$ then for all L, $ML \bowtie NL$ and $M\|L \bowtie N\|L$.*

(ii) *If $M + N \bowtie P + Q$ then one of the following alternatives is true:*
- *$M \bowtie P$ & $N \bowtie Q$ or*
- *$M \bowtie Q$ & $N \bowtie P$ or*
- *$\exists M_0, M_1, N_0, N_1,\ M \bowtie M_0 + M_1$ & $N \bowtie N_0 + N_1$ & $P \bowtie M_0 + N_0$ & $Q \bowtie M_1 + N_1$.*

Proof. Part (i) is straightforward by induction on $M \bowtie N$.

Part (ii) is a consequence of the Church–Rosser property. Indeed, if $M + N \bowtie P + Q$, then there are L and L' such that $M + N \rhd^* L$, $P + Q \rhd^* L'$, and L and L' are equal up to commutativity and associativity of $+$ and $\|$. But any sum of the shape $M + N$ can be reduced only to a sum $M' + N'$ where $M \rhd^* M'$ and $N \rhd^* N'$, and similarly for $P + Q$. The thesis then follows. □

The next lemma connects the relation $\rhd$ to the reduction trees of terms and hence to the reduction relation $\longrightarrow$.

Notation. From now on we abuse notation, writing just $\rhd$ instead of $\rhd^*$ (unless otherwise stated).

LEMMA 2.9. *Let $M \in \Lambda^0_{+\|}$. Then*

(i) $\mathcal{R}(M, 1) = \{M_1, \ldots, M_n\} \Rightarrow M \rhd \sum_{i=1}^n M_i$,

(ii) $\forall b \in bar(M),\ b = \{M_1, \ldots, M_n\} \Rightarrow M \rhd \sum_{i=1}^n M_i$.

Proof. (i) The proof is by induction on $M \in \Lambda^0_{+\|}$.

—If $M \equiv \lambda\chi.M' \in \Lambda^0_{+\|}$ (that is, $FV(M') \subseteq \{\chi\}$) then $M \in \mathsf{TVal}$ and $\mathcal{R}(M, 1) = \{M\}$.

—If $M \equiv PQ$ then $P, Q \in \Lambda^0_{+\|}$. We have some subcases. If $P \equiv \lambda\chi.P'$ and either $\chi \equiv x$ or both $\chi \equiv v$ and $Q \in \mathsf{TVal}$, then

$$\mathcal{R}(PQ, 1) = \{P'[Q/\chi]\} \quad \text{and} \quad PQ \rhd P'[Q/\chi]$$

by $(\beta)'$ or by $(\beta_v)'$.

Suppose that $\mathcal{R}(Q, 1) = \{Q_1, \ldots, Q_k\}$. If $P \equiv \lambda v.P'$ and $Q \in \mathsf{Val} - \mathsf{TVal}$ then

$$\mathcal{R}((\lambda v.P')Q, 1) = \{P'[Q/v]\|(\lambda v.P')Q_i \mid i \leq k\}$$

and

$$(\lambda v.P')Q \rhd \sum_{i=1}^{k} P'[Q/v]\|(\lambda v.P')Q_i$$

by $(\beta_v\|)'$. Otherwise, if $Q \notin \mathsf{Val}$, then

$$\mathcal{R}((\lambda v.P')Q, 1) = \{(\lambda v.P')Q_i \mid i \leq k\} \text{ and } (\lambda v.P')Q \rhd \sum_{i=1}^{k} (\lambda v.P')Q_i$$

by $(\mu_v)'$.

If $P \equiv P_0 \| P_1$ then

$$\mathcal{R}(PQ, 1) = \{P_0 Q \| P_1 Q\} \text{ and } PQ \rhd P_0 Q \| P_1 Q$$

by $(\|_{app})'$.

In all other subcases, $P \notin \mathsf{Val} \cup \mathsf{Par}$. Now let $\mathcal{R}(P, 1) = \{P_1, \ldots, P_h\}$; hence we have $P \rhd \sum_{i=1}^{h} P_i$ by induction hypothesis and

$$PQ \rhd \left(\sum_{i=1}^{h} P_i\right) Q \rhd \sum_{i=1}^{h} (P_i Q).$$

From this the thesis follows since in these cases $\mathcal{R}(PQ, 1) = \{P_i Q \mid i \leq h\}$.

—If $M \equiv P + Q$ then $P, Q \in \Lambda^0_{+\|}$ and the case is trivial since $\mathcal{R}(P + Q, 1) = \{P, Q\}$.

—If $M \equiv P \| Q$ then $P, Q \in \Lambda^0_{+\|}$. Again suppose that $\mathcal{R}(P, 1) = \{P_1, \ldots, P_h\}$ and $\mathcal{R}(Q, 1) = \{Q_1, \ldots, Q_k\}$. Then we have

$$P \| Q \rhd \left(\sum_{i=1}^{h} P_i\right) \| \left(\sum_{j=1}^{k} Q_j\right) \rhd \sum_{i=1}^{h} \sum_{j=1}^{k} (P_i \| Q_j)$$

by induction hypothesis and clauses $(\|+)'$ and $(\|)'$. So we are done since $\mathcal{R}(PQ, 1) = \{P_i \| Q_j \mid i \leq h, j \leq k\}$.

(ii) This proof is by induction on the height h of the bar b. If $h = 0$ then the thesis is trivial. Otherwise $tree(M)$ has the root node labeled by M and $tree(M_1), \ldots, tree(M_n)$ as its immediate subtrees, where $\{M_1, \ldots, M_n\} = \mathcal{R}(M, 1)$. Because of $h \neq 0$ we have that the root is not in b (recall that a bar intersects each maximal path of $tree(M)$ in exactly one node). It follows that for all $i \leq n$ there exists $b_i \in bar(M_i)$ such that $b = b_1 \cup \cdots \cup b_n$. But the height of each b_i with respect to $tree(M_i)$ has to be less than h, so that $M_i \rhd \sum\{M_i' \mid M_i' \in b_i\}$ by induction hypothesis. So the thesis follows from (i) of this lemma. □

Part (ii) of Lemma 2.9 implies $M \rhd \sum \mathcal{R}(M, k)$ for all k. Moreover, it implies that if $M \longrightarrow^* N$ then either $M \rhd N$ or $M \rhd N + L$ for some L.

Observe that the implications in Lemma 2.9 cannot be reversed. This is due to the more permissive clause $(\|)'$ of Definition 2.6. Indeed, if, e.g., $M \equiv (\lambda x.xxx)(\lambda x.xxx)$ then there exists an infinite reduction $M \equiv M_0 \longrightarrow M_1 \longrightarrow \cdots$ where each M_i is an application and $M_i \not\equiv M_{i+1}$ for all i. Now the unique branch of $tree(M_0 \| M_1)$ is the infinite one: $M_0 \| M_1 \longrightarrow M_1 \| M_2 \longrightarrow \cdots$. But $M_0 \| M_1 \rhd M_i \| M_i$ for all $i \geq 1$, while for all $b \in bar(M)$, $b \neq \{M_i \| M_i\}$.

COROLLARY 2.10. *If $N \in \Lambda^0_{+\|}$ and $N \Downarrow$, then there exist $V_1, \ldots, V_n \in \mathsf{Val}$ such that*

(i) $N \rhd \sum_{i=1}^{n} V_i$;

(ii) $\forall (\lambda v.M) \in \Lambda^0_{+\|}$, $(\lambda v.M)N \rhd \sum_{i=1}^{n} (\lambda v.M)V_i$.

Proof. If $N \Downarrow$ then there exists a bar of values $\{V_1, \ldots, V_n\} \in bar(N)$ such that each V_i is the first value that is met starting from the root through a maximal path in $tree(N)$. That is, no value occurs in the path from N to V_i. By (ii) of Lemma 2.9 $N \rhd \sum_{i=1}^{n} V_i$. On the other hand $\{(\lambda v.M)V_i \mid i \leq n\} \in bar((\lambda v.M)N)$ because of rule (μ_v). Then the thesis follows by (ii) of Lemma 2.9. □

LEMMA 2.11. *Let $M, N, V \in \Lambda^0_{+\|}$.*

(i) $(\lambda v.M)V \Downarrow \ \& \ V \in \mathsf{Val} \ \Rightarrow \exists V_1, \ldots, V_n \in \mathsf{Val},\ V \rhd \sum_{i=1}^n V_i \ \& \ \forall i \leq n,\ M[V_i/v]\Downarrow$.

(ii) $(\lambda v.M)N\Downarrow \Rightarrow \exists V_1, \ldots, V_n \in \mathsf{Val},\ N \rhd \sum_{i=1}^n V_i \ \& \ \forall i \leq n,\ M[V_i/v]\Downarrow$.

Proof. (i) If $V \in \mathsf{TVal}$ then $\mathcal{R}((\lambda v.M)V, 1) = \{M[V/v]\}$, so the hypothesis implies that $M[V/v]\Downarrow$. Otherwise $V \in \mathsf{Val} - \mathsf{TVal}$. By definition, $(\lambda v.M)V \Downarrow_k$ for some $k > 0$, and we make induction on k. Suppose that $\mathcal{R}(V, 1) = \{V_1, \ldots, V_n\}$, so that

$$\mathcal{R}((\lambda v.M)V, 1) = \{M[V/v] \| (\lambda v.M)V_i \mid i \leq n\}.$$

If $k = 1$ then, since for all i, $(\lambda v.M)V_i$ is an application (that is, it is not a value), we have $M[V/v] \in \mathsf{Val}$, so that $M[V/v] \Downarrow$. If $k > 1$ and $M[V/v] \Uparrow$ (otherwise the thesis is immediate), then for all $i \leq n$, $(\lambda v.M)V_i \Downarrow_{k-1}$. By induction there are $V_{i,1}, \ldots, V_{i,n_i} \in \mathsf{Val}$ such that $V_i \rhd \sum_{j=1}^{n_i} V_{i,j}$ and $M[V_{i,j}/v] \Downarrow$ for all i and j. The thesis now follows since $V \rhd \sum_{i=1}^n V_i$ by (i) of Lemma 2.9.

(ii) If $N \Uparrow$ then for all $k \geq 0$ there exists $N' \in \mathcal{R}(N, k)$ such that $(\lambda v.M)N' \in \mathcal{R}((\lambda v.M)N, k)$ and rules (β_v) and $(\beta_v\|)$ cannot be applied to $(\lambda v.M)N'$. This implies that $(\lambda v.M)N \Uparrow$. By hypothesis and by contraposition we have that $N\Downarrow$. By Corollary 2.10 there exist values $V_1, \ldots, V_n$ such that $N \rhd \sum_{i=1}^n V_i$ and $(\lambda v.M)N \rhd \sum_{i=1}^n (\lambda v.M)V_i$. Moreover, by the proof of the same corollary, $\{(\lambda v.M)V_i \mid i \leq n\} \in bar((\lambda v.M)N)$, so that by hypothesis $(\lambda v.M)V_i \Downarrow$ for all $i \leq n$. Now, by part (i) of this lemma, for each i there are $V_{i,1}, \ldots, V_{i,n_i} \in \mathsf{Val}$ such that $V_i \rhd \sum_{j=1}^{n_i} V_{i,j}$ and $M[V_{i,j}/v]\Downarrow$, and the thesis follows. □

THEOREM 2.12. *Let* $M, N \in \Lambda^0_{+\|}$. *Then*

(i) $[M \rhd N \ \& \ N\Downarrow] \ \Rightarrow M\Downarrow$;

(ii) $M \bowtie N \Rightarrow [M\Downarrow \Leftrightarrow N\Downarrow]$.

Proof. (i) In this proof we must distinguish between $\rhd$ and $\rhd^*$. Clearly, if we can prove the statement for $\rhd$, the same thesis holds for $\rhd^*$. As a matter of fact we prove, by induction on the definition of $\rhd$, the stronger statement

$$M \rhd N \Rightarrow \forall \vec{L},\ [N\vec{L}\Downarrow \ \Rightarrow M\vec{L}\Downarrow],$$

from which the thesis follows, taking the empty vector.

—If $M \rhd N$ thanks to $(\beta)'$, $(\beta_v)'$, $(\beta_v\|)'$, $(\mu_v)'$, $(+_{app})'$, or $(\|_{app})'$ (see Definition 2.6), then M is always an application and $N \equiv \sum_{i=1}^n M_i$ where $\mathcal{R}(M, 1) = \{M_1, \ldots, M_n\}$ (where the multiset $\mathcal{R}(M, 1)$ is ordered in such a way that it matches the shape of N). By this fact and rules (ν) and $(+)$ we have that

$$\{M_i\vec{L} \mid 1 \leq i \leq n\} \in bar(N\vec{L}) \cap bar(M\vec{L}).$$

Now $N\vec{L}\Downarrow$ implies that $M_i\vec{L}\Downarrow$ for all i $(1 \leq i \leq n)$, so that $M\vec{L}\Downarrow$ follows.

—Clause $(\|+)'$. Then $M \equiv (P + Q)\|R$ and $N \equiv P\|R + Q\|R$. Now

$$\begin{aligned}(P\|R + Q\|R)\vec{L}\Downarrow &\Rightarrow (P\vec{L}\Downarrow \ \& \ Q\vec{L}\Downarrow) \text{ or } R\vec{L}\Downarrow \\ &\Rightarrow (P\vec{L}\Downarrow \text{ or } R\vec{L}\Downarrow) \ \& \ (Q\vec{L}\Downarrow \text{ or } R\vec{L}\Downarrow) \\ &\Rightarrow ((P + Q)\|R)\vec{L}\Downarrow\end{aligned}$$

by rule (ν) and the remark after Definition 2.5.

—Clause $(\nu)'$. Then $M \equiv PQ$, $N \equiv P'Q$, and $P \rhd P'$. In this case $P'Q\vec{L}\Downarrow$ implies $PQ\vec{L}\Downarrow$ immediately by induction, taking the vector $Q\vec{L}$.

—Clause $(+)'$. Then $M \equiv P + Q$, $N \equiv P' + Q$, and $P \rhd P'$. Now

$$\begin{array}{rcll}(P' + Q)\vec{L}\Downarrow & \Rightarrow & P'\vec{L}\Downarrow \ \& \ Q\vec{L}\Downarrow & \\ & \Rightarrow & P\vec{L}\Downarrow \ \& \ Q\vec{L}\Downarrow & \text{by induction} \\ & \Rightarrow & (P+Q)\vec{L}\Downarrow . & \end{array}$$

—Clause $(\|)'$. Similar to the case of clause $(+)'$, where "or" replaces "&."

—For clauses $(+_c)'$, $(\|_c)'$, $(+_{ass})'$, and $(\|_{ass})'$ the proofs are similar to those of $(+)'$ and $(\|)'$.

(ii) M converging implies that there are values $V_1, \ldots, V_n$ such that $M \rhd \sum_{i=1}^n V_i$ by Corollary 2.10(i). Therefore, $N \bowtie \sum_{i=1}^n V_i$. By the Church–Rosser property of $\rhd$ there is an L such that $N \rhd L$ and $\sum_{i=1}^n V_i \rhd L$. But $\sum_{i=1}^n V_i \rhd L$ implies that L is a sum of values and therefore L must converge. We conclude that N converges by (i). □

Based on the convergence predicate the following definition adapts to the present setting the notion of contextual theories. This notion stems from [52] and it is widely used in, e.g., [15], for the classical theory of solvability and in [6], [19], and [55], where it is shown to be equivalent to applicative bisimulation.

The idea is that two terms are operationally equivalent if and only if in all contexts they exhibit the same behavior with respect to some observable properties. Here convergence is the only observable property; hence we can make the following definition.

DEFINITION 2.13. *Let $M, N \in \Lambda_{+\|}$. Then*

(i) $M \sqsubseteq^{\mathcal{O}} N \Leftrightarrow \forall C[\,],\ C[M]\Downarrow \Rightarrow C[N]\Downarrow$, *where* $C[M], C[N] \in \Lambda^0_{+\|}$;

(ii) $\simeq^{\mathcal{O}} = \sqsubseteq^{\mathcal{O}} \cap \sqsupseteq^{\mathcal{O}}$.

3. A logical presentation. To obtain a logical presentation of the semantics of the calculus we follow the paradigm of Leibniz which identifies objects with sets of their properties. This received an elegant mathematical treatment thanks to works like [63] and [5] and, especially in the case of type-free calculi, it is naturally formalized in suitable extensions of a Curry-type assignment system like the intersection-type discipline considered in [16].

In the present case we use a more expressive system which allows for disjunctive types. We call them union types since they differ from coproducts in Church-typed λ-calculi in much the same way as intersection differs from cartesian product. See [14] for a study of this discipline in the case of classical λ-calculus.

3.1. The set of types and its preorder. The type syntax is as follows:

$$\sigma ::= \omega \mid \sigma \to \sigma \mid \sigma \wedge \sigma \mid \sigma \vee \sigma,$$

and we call *Type* the resulting set. In writing types, we assume that $\wedge$ and $\vee$ take precedence over $\to$.

The choice of the preorder on types is crucial, since it will be used in a subtyping rule in subsection 3.2 and it will determine the structure of the set of filters in section 4.

DEFINITION 3.1. *Let $\sigma \leq \tau$ be the smallest preorder over types such that*

(i) *$\langle Type, \leq\rangle$ is a distributive lattice, in which $\wedge$ is the meet, $\vee$ is the join, and ω is the top*;

(ii) *the arrow satisfies*

(a) $\sigma \to \omega \leq \omega \to \omega$,

(b) $(\sigma \to \rho) \wedge (\sigma \to \tau) \leq \sigma \to \rho \wedge \tau$,

(c) $\sigma \geq \sigma', \tau \leq \tau' \Rightarrow \sigma \to \tau \leq \sigma' \to \tau'$.

Following [30], by lattice we mean a poset in which every finite nonempty subset has a meet and a join. According to this definition, there are lattices without bottom (like the present one).

We write $\sigma = \tau$ for "$\sigma \leq \tau$ and $\tau \leq \sigma$." Note that, if $\sigma \neq \omega$ then $\sigma \leq \omega \to \omega$.

NOTATION 3.2. Let $\omega^0 \to \omega = \omega$, $\omega^{n+1} \to \omega = \omega \to \omega^n \to \omega$.

The types $\omega^n \to \omega$ for suitable n are "better than" all other types, as shown in the following proposition.

PROPOSITION 3.3. *For all σ, there exists n such that $\omega^n \to \omega \leq \sigma$,*

Proof. By induction on the structure of σ,

$\sigma \equiv \omega$: This proof is trivial.

$\sigma \equiv \sigma_1 \to \sigma_2$: By the induction hypothesis, $\exists n$, $\omega^n \to \omega \leq \sigma_2$, hence $\omega^{n+1} \to \omega \leq \sigma$, by Definition 3.1(ii(c)), using $\sigma_1 \leq \omega$.

$\sigma \equiv \sigma_1 \wedge \sigma_2$: By the induction hypothesis, $\exists n_i$, $\omega^{n_i} \to \omega \leq \sigma_i$ $(i = 1, 2)$. Let $n = \max(n_1, n_2)$. Then $\omega^n \to \omega \leq \sigma$, since $\omega^n \to \omega \leq \omega^{n_i} \to \omega$ $(i = 1, 2)$ and $\sigma_1 \wedge \sigma_2$ is the meet of σ_1 and σ_2.

$\sigma \equiv \sigma_1 \vee \sigma_2$: Recall that $\sigma_1 \wedge \sigma_2 \leq \sigma_1 \vee \sigma_2$, and then proceed as in the previous case. □

We need some properties of the $\leq$ relation, whose proof requires a stratification of $Type$.

DEFINITION 3.4 (stratification of $Type$). *Let us define three subsets T_0, T_1, T_2 of $Type$ recursively:*

$\omega \to \omega \in T_0$;

$\omega \in T_2$;

$\sigma \in T_2, \tau \in T_1 \Rightarrow \sigma \to \tau \in T_0$;

$n \geq 1, \sigma_1, \ldots, \sigma_n \in T_0 \Rightarrow \sigma_1 \vee \cdots \vee \sigma_n \in T_1$;

$n \geq 1, \sigma_1, \ldots, \sigma_n \in T_1 \Rightarrow \sigma_1 \wedge \cdots \wedge \sigma_n \in T_2$.

To rephrase the previous definition, we consider types in conjunctive normal form, that is, conjunctions of disjunctions of arrows, ω being the empty conjunction.

REMARK 3.5. Notice that the set T_2, when restricted to types without $\vee$ occurrences, is similar to the set of normal type schemes of [36] and to the set of strict types of [12]. The difference is that in those papers types were constructed out of type variables and the type $\omega \to \omega$ was not allowed. Normal-type schemes were introduced in [36] as a technical tool to prove properties analogous to those stated in Lemma 3.9. Instead, strict types have been introduced with a different preorder to obtain a syntax-directed type assignment system [12], [13].

Taking $n = 1$ in the clauses above, one sees that $T_0 \subseteq T_1 \subseteq T_2$, and such inclusions are clearly proper.

Over each of these sets we introduce a preorder.

DEFINITION 3.6. *$\leq_i \subseteq T_i \times T_i$ is the least preorder such that*

$(\leq_0)$: $\sigma \leq_0 \tau \;\Leftrightarrow\; \tau \equiv \omega \to \omega$ *or* $\sigma \equiv \sigma' \to \sigma'', \tau \equiv \tau' \to \tau''$ *and* $\tau' \leq_2 \sigma'$ *and* $\sigma'' \leq_1 \tau''$;

$(\leq_1)$: $\sigma_1 \vee \cdots \vee \sigma_n \leq_1 \tau_1 \vee \cdots \vee \tau_m \;\Leftrightarrow\; \forall i \leq n\ \exists j \leq m,\ \sigma_i \leq_0 \tau_j$;

$(\leq_2)$: $\sigma \leq_2 \tau \;\Leftrightarrow\; \tau \equiv \omega$ *or* $\sigma \equiv \sigma_1 \wedge \cdots \wedge \sigma_n$, $\tau \equiv \tau_1 \wedge \cdots \wedge \tau_m$ *and* $\forall j \leq m\ \exists i \leq n,\ \sigma_i \leq_1 \tau_j$.

Really, for each type in $Type$, we can find an equivalent type in T_2; therefore, we introduce a map which associates to each type its equivalent type in T_2.

DEFINITION 3.7. *Let $* : Type \to T_2$ be defined by*

$$\omega^* = \omega;$$

$$(\sigma \to \tau)^* = \begin{cases} \bigwedge_{i\in I}(\sigma^* \to \tau_i) & \text{if } \tau^* \equiv \bigwedge_{i\in I} \tau_i \text{ and } \tau^* \not\equiv \omega, \\ \omega \to \omega & \text{otherwise;} \end{cases}$$

$$(\sigma \vee \tau)^* = \begin{cases} \bigwedge_{i\in I}\bigwedge_{j\in J}(\sigma_i \vee \tau_j) & \text{if } \sigma^* \equiv \bigwedge_{i\in I} \sigma_i,\ \sigma^* \not\equiv \omega \\ & \text{and } \tau^* \equiv \bigwedge_{j\in J} \tau_j,\ \tau^* \not\equiv \omega, \\ \omega & \text{otherwise;} \end{cases}$$

$$(\sigma \wedge \tau)^* = \begin{cases} \sigma^* & \text{if } \tau^* \equiv \omega, \\ \tau^* & \text{if } \sigma^* \equiv \omega, \\ \sigma^* \wedge \tau^* & \text{otherwise.} \end{cases}$$

PROPOSITION 3.8. *For all $\sigma, \tau \in Type$,*

(i) $\sigma = \sigma^*$;

(ii) $\sigma, \tau \in T_i, \sigma \leq_i \tau \Rightarrow \sigma \leq \tau$ *for* $i = 0, 1, 2$;

(iii) $\sigma \leq \tau \Rightarrow \sigma^* \leq_2 \tau^*$.

Proof. (i) is proved by induction on the definition of the map $(\cdot)^*$. To see (ii), use an induction on the definition of $\leq_i$. Finally, to prove (iii), it suffices (by (i)) to show that $\sigma^* \leq \tau^*$ implies $\sigma^* \leq_2 \tau^*$, which can be proved by induction on any standard axiomatic presentation of $\leq$. □

LEMMA 3.9.

(i) $\mu \wedge \nu \leq \sigma \to \tau \ \&\ \mu \neq \omega \ \&\ \nu \neq \omega \Rightarrow \exists \tau_1, \tau_2,\ \tau = \tau_1 \wedge \tau_2 \ \&\ \mu \leq \sigma \to \tau_1 \ \&\ \nu \leq \sigma \to \tau_2$;

(ii) $\bigwedge_{i\in I}(\mu_i \to \nu_i) \leq \sigma \to \tau \ \&\ \tau \neq \omega \Rightarrow \exists J \subseteq I,\ \sigma \leq \bigwedge_{j\in J} \mu_j \ \&\ \bigwedge_{j\in J} \nu_j \leq \tau$.

Proof. (i) Let

$$(\mu \wedge \nu)^* = \bigwedge_{i\in I} \mu_i \wedge \bigwedge_{j\in J} \nu_j \text{ and } (\sigma \to \tau)^* = \bigwedge_{k\in K} (\sigma^* \to \pi_k),$$

assuming $\mu^* = \bigwedge_{i\in I} \mu_i, \nu^* = \bigwedge_{j\in J} \nu_j$ and $\tau^* = \bigwedge_{k\in K} \pi_k$. Using Proposition 3.8(i), (ii), (iii) and the definition of $\leq_2$, we have that

$$\forall k,\ (\exists i,\ \mu_i \leq_1 \sigma^* \to \pi_k) \text{ or } (\exists j,\ \nu_j \leq_1 \sigma^* \to \pi_k).$$

Therefore we can choose τ_1 as the intersection of the π_k which satisfy the first inequality and τ_2 as the intersection of the remaining π_k. If one of these intersections is empty, we choose ω for the corresponding τ_i $(i = 1, 2)$.

(ii) Let $\nu_i^* = \bigwedge_{l\in L} \nu_{i,l}$ (where L depends on i) and $\tau^* = \bigwedge_{k\in K} \tau_k$. Then

$$\bigwedge_{i\in I}(\mu_i \to \nu_i) \leq \sigma \to \tau \Rightarrow \bigwedge_{i\in I}\bigwedge_{l\in L}(\mu_i^* \to \nu_{i,l}) \leq_2 \bigwedge_{k\in K}(\sigma^* \to \tau_k).$$

It follows that

$$\forall k\ \exists i, l,\ \mu_i^* \to \nu_{i,l} \leq_1 \sigma^* \to \tau_k,$$

which in this case is equivalent to

$$\forall k\ \exists i, l,\ \mu_i^* \to \nu_{i,l} \leq_0 \sigma^* \to \tau_k,$$

and hence

$$\forall k\ \exists i, l,\ \sigma^* \leq_2 \mu_i^* \ \&\ \nu_{i,l} \leq_1 \tau_k.$$

So we can conclude that

$$\forall k\ \exists i,\ \sigma \leq \mu_i\ \&\ \bigwedge_{l\in L} \nu_{i,l} \leq \tau_k.$$

Taking J as the set of all i which satisfy these inequalities for some $k \in K$, we are done. □

REMARK 3.10. Notice that Lemma 3.9 cannot be trivially satisfied by choosing $\tau_1 = \tau_2 = \tau$. In fact, in general, $\mu \wedge \nu \leq \sigma \to \tau$ does not imply $\mu \leq \sigma \to \tau$. For a counterexample, take $\mu = \sigma = \tau = \omega \to \omega$ and $\nu = \sigma \to \tau$.

A type σ is *join-irreducible* or *coprime* if and only if

$$\sigma \leq \tau \vee \rho \Rightarrow \sigma \leq \tau \text{ or } \sigma \leq \rho$$

for any τ, ρ. Let $CType$ be the set of coprime types different from ω. Observe that, because of distributivity, coprime types are closed under $\wedge$. Being $\langle Type, \leq\rangle$, the free distributive lattice satisfying the arrow axioms, each type is the join of a finite number of coprime types. To see this, it suffices to define the following mapping $\Theta : Type \to \mathcal{P}(CType)$:

$$\begin{array}{lcl}
\Theta(\omega) & = & \{\omega\},\\
\Theta(\sigma \to \tau) & = & \{\sigma \to \tau\},\\
\Theta(\sigma \wedge \tau) & = & \{\sigma' \wedge \tau' \mid \sigma' \in \Theta(\sigma)\ \&\ \tau' \in \Theta(\tau)\},\\
\Theta(\sigma \vee \tau) & = & \Theta(\sigma) \cup \Theta(\tau).
\end{array}$$

If $\Theta(\sigma) = \{\sigma_1, \ldots, \sigma_n\}$, it is easy to verify that σ_i is join-irreducible for each i and $\sigma = \sigma_1 \vee \cdots \vee \sigma_n$.

3.2. The type assignment system. In this subsection we introduce our type assignment system $\mathcal{L}$. We start with the notion of basis. We state that only coprime types different from ω can be assumed for call-by-value variables. This restriction is justified by the correspondence between total values and coprime types (see Theorem 3.15(ii)).

DEFINITION 3.11. *A* basis $\Gamma : (\mathsf{Vn} \to Type) \cap (\mathsf{Vv} \to CType)$ *is a mapping such that* $\Gamma(x) = \omega$ *for all* x *but a finite subset of* Vn*, and* $\Gamma(v) = \omega \to \omega$ *for all* v *but a finite subset of* Vv.

To each basis Γ we associate the finite set

$$Dom(\Gamma) = \{x \in \mathsf{Vn} \mid \Gamma(x) \neq \omega\} \cup \{v \in \mathsf{Vv} \mid \Gamma(v) \neq \omega \to \omega\}.$$

The notation $\Gamma, \chi : \sigma$ is shorthand for the function $\Gamma'(\chi') = \sigma$ if $\chi' \equiv \chi$; $\Gamma(\chi')$ otherwise. To meet a common practice we shall sometimes identify Γ with the (finite) set of judgments $\{\chi{:}\sigma \mid \chi \in Dom(\Gamma)\ \&\ \Gamma(\chi) = \sigma\}$ and write $\chi{:}\sigma \in \Gamma$.

DEFINITION 3.12. *The axioms and rules of the assignment system* $\mathcal{L}$ *are as follows:*

$$(\mathrm{Ax})\quad \Gamma \vdash \chi : \Gamma(\chi), \qquad (\omega)\quad \Gamma \vdash M : \omega,$$

$$(\to \mathrm{I}_n)\quad \frac{\Gamma, x:\sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau}, \qquad (\to \mathrm{I}_v)\quad \frac{\Gamma, v:\sigma' \vdash M : \tau \quad \forall \sigma' \in \Theta(\sigma)}{\Gamma \vdash \lambda v.M : \sigma \to \tau},$$

$$(\to \mathrm{E})\quad \frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau},$$

$$(\wedge \mathrm{I})\quad \frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash M : \tau}{\Gamma \vdash M : \sigma \wedge \tau}, \qquad (\leq)\quad \frac{\Gamma \vdash M : \sigma \quad \sigma \leq \tau}{\Gamma \vdash M : \tau},$$

$$(+\mathrm{I})\quad \frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash M + N : \sigma \vee \tau}, \qquad (\| \mathrm{I})\quad \frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash M \| N : \sigma \wedge \tau}.$$

$\Gamma \vdash_{\mathcal{L}} M : \sigma$ *abbreviates "*$\Gamma \vdash M : \sigma$ *is derivable in* $\mathcal{L}$*."*

To help the understanding of rule $(\to \mathrm{I}_v)$, we consider the following example. Let W_1, W_2 be total values such that $\vdash_{\mathcal{L}} W_i{:}\sigma_i$ $(i = 1, 2)$ for some coprime types σ_1, σ_2. Clearly, this implies $\vdash_{\mathcal{L}} W_1 + W_2{:}\sigma_1 \vee \sigma_2$ by rule $(+\mathrm{I})$. Consider $(\lambda v.M)(W_1 + W_2)$: it reduces to $M[W_1/v]$ and $M[W_2/v]$. Therefore $v{:}\sigma_i \vdash_{\mathcal{L}} M{:}\tau$ for $i = 1, 2$ suffices to assure that $(\lambda v.M)$ has type $\sigma_1 \vee \sigma_2 \to \tau$. The real justification of this rule is that it implies the completeness of the type assignment (Theorem 4.11) and the full abstraction of the filter model (Theorem 5.11).

We shall write $\Gamma \leq \Gamma'$ if, $\forall \chi,\ \Gamma(\chi) \leq \Gamma'(\chi)$; in this case it is easy to verify that, if $\Gamma' \vdash_{\mathcal{L}} M : \sigma$, then $\Gamma \vdash_{\mathcal{L}} M : \sigma$ for any M and σ.

The system $\mathcal{L}$ enjoys structural properties which can be shown by simple inductions on derivations.

THEOREM 3.13 (derivability properties of system $\mathcal{L}$).

(i) $\Gamma \vdash_{\mathcal{L}} \chi : \tau \ \Leftrightarrow\ \Gamma(\chi) \leq \tau$;

(ii) $\Gamma \vdash_{\mathcal{L}} \lambda\chi.M : \rho \ \Leftrightarrow$
$\exists n,\ \sigma_1, \ldots, \sigma_n, \tau_1, \ldots \tau_n.$
$(\forall i \leq n,\ \Gamma \vdash_{\mathcal{L}} \lambda\chi.M : \sigma_i \to \tau_i)\ \&\ \bigwedge_{i=1}^{n}(\sigma_i \to \tau_i) \leq \rho$;

(iii) $\Gamma \vdash_{\mathcal{L}} \lambda x.M : \sigma \to \tau \ \Leftrightarrow\ \Gamma, x:\sigma \vdash_{\mathcal{L}} M : \tau$;

(iv) $\Gamma \vdash_{\mathcal{L}} \lambda v.M : \sigma \to \tau\ \&\ \sigma \neq \omega \ \Leftrightarrow\ \forall \sigma' \in \Theta(\sigma),\ \Gamma, v:\sigma' \vdash_{\mathcal{L}} M : \tau$;

(v) $\Gamma \vdash_{\mathcal{L}} \lambda v.M : \sigma \to \tau\ \&\ \sigma = \omega \Rightarrow \tau = \omega$;

(vi) $\Gamma \vdash_{\mathcal{L}} MN : \tau\ \&\ \tau \neq \omega \ \Leftrightarrow\ \exists \sigma,\ \Gamma \vdash_{\mathcal{L}} M : \sigma \to \tau\ \&\ \Gamma \vdash_{\mathcal{L}} N : \sigma$;

(vii) $\Gamma \vdash_{\mathcal{L}} M + N : \sigma \ \Leftrightarrow\ \Gamma \vdash_{\mathcal{L}} M : \sigma\ \&\ \Gamma \vdash_{\mathcal{L}} N : \sigma$;

(viii) $\Gamma \vdash_{\mathcal{L}} M \| N : \tau \ \Leftrightarrow\ \exists \sigma, \sigma',\ \Gamma \vdash_{\mathcal{L}} M : \sigma\ \&\ \Gamma \vdash_{\mathcal{L}} N : \sigma'\ \&\ \sigma \wedge \sigma' \leq \tau$.

Proof. We consider only the interesting cases.

(ii) Given a derivation of $\Gamma \vdash \lambda\chi.M{:}\rho$, let

$$\Gamma \vdash \lambda\chi.M{:}\sigma_1 \to \tau_1, \ldots, \Gamma \vdash \lambda\chi.M{:}\sigma_n \to \tau_n$$

be all the statements in this deduction on which $\Gamma \vdash \lambda\chi.M{:}\rho$ depends and which are conclusions of rule $(\to \mathrm{I}_n)$ or of rule $(\to \mathrm{I}_v)$. Then

$$(\sigma_1 \to \tau_1) \wedge \cdots \wedge (\sigma_n \to \tau_n) \leq \rho.$$

(iii) If $\tau = \omega$ it is trivial. Otherwise let $\sigma_1, \ldots, \sigma_n, \tau_1, \ldots, \tau_n$ be as in the proof of (ii), where ρ has been replaced by $\sigma \to \tau$. Then

$$(\sigma_1 \to \tau_1) \wedge \cdots \wedge (\sigma_n \to \tau_n) \leq \sigma \to \tau,$$

which implies, by Lemma 3.9(ii),

$$\exists J \subseteq \{1, \ldots, n\},\ \sigma \leq \bigwedge_{j \in J} \sigma_j\ \&\ \bigwedge_{j \in J} \tau_j \leq \tau.$$

Moreover, $\Gamma, x{:}\sigma_i \vdash_{\mathcal{L}} M : \tau_i$ for $1 \le i \le n$, so that one can conclude $\Gamma, x{:}\sigma \vdash_{\mathcal{L}} M{:}\tau$.

(iv) Let $\sigma_1, \dots, \sigma_n, \tau_1, \dots, \tau_n$ be as in the proof of (ii), where ρ has been replaced by $\sigma \to \tau$. As in case (iii) we have

$$\exists J \subseteq \{1, \dots, n\},\ \sigma \le \bigwedge_{j \in J} \sigma_j \ \&\ \bigwedge_{j \in J} \tau_j \le \tau.$$

Moreover, $\Gamma, v{:}\sigma'_i \vdash_{\mathcal{L}} M : \tau_i$ for all $\sigma'_i \in \Theta(\sigma_i)$ and for $1 \le i \le n$. $\sigma \le \sigma_j$ implies, $\forall \sigma' \in \Theta(\sigma)$, $\exists \sigma'_j \in \Theta(\sigma_j)$ such that $\sigma' \le \sigma'_j$ by the definition of coprimality. So we can conclude, $\forall \sigma' \in \Theta(\sigma)$, $\Gamma, x{:}\sigma' \vdash_{\mathcal{L}} M{:}\tau$.

(v) We assume ad absurdum that $\tau \ne \omega$. Then, if $\sigma_1, \dots, \sigma_n, \tau_1, \dots, \tau_n$ and J are as in (iv), we would have $\sigma_j = \omega$ for all $j \in J$, and this is impossible according to our definition of basis.

(vii) Again, given a deduction of $\Gamma \vdash M + N{:}\sigma$, let

$$\Gamma \vdash M + N{:}\sigma_1, \dots, \Gamma \vdash M + N{:}\sigma_n$$

be all the statements in this deduction on which $\Gamma \vdash M + N{:}\sigma$ depends and which are conclusions of rule (+I). Then $\sigma_1 \wedge \cdots \wedge \sigma_n \le \sigma$ and there are μ_i, ν_i such that $\sigma_i = \mu_i \vee \nu_i$, $\Gamma \vdash_{\mathcal{L}} M{:}\mu_i$, $\Gamma \vdash_{\mathcal{L}} N{:}\nu_i$, for $1 \le i \le n$. So we can deduce $\Gamma \vdash M{:}\sigma$ and $\Gamma \vdash N{:}\sigma$ using ($\wedge$ I) and ($\le$).

(viii) Finally, given a deduction of $\Gamma \vdash M \| N{:}\tau$, let

$$\Gamma \vdash M \| N{:}\sigma_1, \dots, \Gamma \vdash M \| N{:}\sigma_n$$

be all the statements in this deduction on which $\Gamma \vdash M \| N{:}\tau$ depends and which are conclusions of rule ($\|$I). Then $\sigma_1 \wedge \cdots \wedge \sigma_n \le \tau$ and there are μ_i, ν_i such that $\sigma_i = \mu_i \wedge \nu_i$, $\Gamma \vdash_{\mathcal{L}} M{:}\mu_i$, $\Gamma \vdash_{\mathcal{L}} N{:}\nu_i$, for $1 \le i \le n$. Then we can choose $\sigma = \bigwedge_{i \le n} \mu_i$ and $\sigma' = \bigwedge_{i \le n} \nu_i$. In fact, $\sigma \wedge \sigma' \le \tau$ and we can derive $\Gamma \vdash_{\mathcal{L}} M{:}\sigma$ and $\Gamma \vdash N{:}\sigma'$ using ($\wedge$ I). □

As an immediate consequence of Theorem 3.13(iv) we have the co-additivity of call-by-value abstraction (i.e., finite meets are preserved).

COROLLARY 3.14. $\Gamma \vdash_{\mathcal{L}} \lambda v.M : (\sigma \to \rho) \wedge (\tau \to \rho) \ \Rightarrow\ \Gamma \vdash_{\mathcal{L}} \lambda v.M : \sigma \vee \tau \to \rho$.

We show how types characterize partial values and total values.

THEOREM 3.15 (characterization of values).

(i) $V \in \mathsf{Val} \Rightarrow \vdash_{\mathcal{L}} V : \omega \to \omega$;

(ii) $W \in \mathsf{TVal}\ \&\ \Gamma \vdash_{\mathcal{L}} W : \sigma \Rightarrow \exists \sigma' \in \Theta(\sigma),\ \Gamma \vdash_{\mathcal{L}} W : \sigma'$.

Proof. (i) The proof is by induction on the definition of values. If $V \equiv v \in \mathsf{Vv}$ then $\Gamma(v) = \omega \to \omega$, since $Dom(\Gamma) = \emptyset$, and the thesis follows by (Ax). If $V \equiv \lambda x.M$ then $\Gamma, x : \omega \vdash M : \omega$ is derivable by rule (ω), hence the thesis using ($\to \mathrm{I}_n$). If $V \equiv \lambda v.M$ we do the same as before, but assuming $v : \omega \to \omega$. The thesis follows using ($\to \mathrm{I}_v$) and ($\le$). Finally, if $V \equiv V' \| M$ or $M \| V'$ the thesis follows by induction using ($\|$) and ($\le$).

(ii) The proof is by induction on the definition of total values. If $W \equiv v$ then the thesis follows by Theorem 3.13(i) and the definition of basis. If $W \equiv \lambda\chi.M$ then, noting that each arrow type is coprime, the thesis follows from Theorem 3.13(ii) and the closure of coprime types under $\wedge$. Finally, if $W \equiv W' \| W''$, by Theorem 3.13(viii) there exist μ and ν such that $\Gamma \vdash_{\mathcal{L}} W' : \mu$, $\Gamma \vdash_{\mathcal{L}} W'' : \nu$, and $\mu \wedge \nu \le \sigma$; now by induction there are $\mu' \in \Theta(\mu), \nu' \in \Theta(\nu)$ such that $\Gamma \vdash_{\mathcal{L}} W' : \mu'$ and $\Gamma \vdash_{\mathcal{L}} W'' : \nu'$.

Since coprime types are closed under $\wedge$, $\mu' \wedge \nu' \leq \sigma$ implies that there is $\sigma' \in \Theta(\sigma)$ such that $\mu' \wedge \nu' \leq \sigma'$. □

The following lemma states the substitution properties of terms.

LEMMA 3.16.

(i) $\Gamma \vdash_{\mathcal{L}} M[N/x] : \tau \Leftrightarrow \exists \sigma,\ \Gamma, x : \sigma \vdash_{\mathcal{L}} M : \tau \ \&\ \Gamma \vdash_{\mathcal{L}} N : \sigma$;

(ii) $\Gamma \vdash_{\mathcal{L}} M[V/v] : \tau \ \&\ V \in \mathsf{Val} \Rightarrow \exists \sigma\ \forall \sigma' \in \Theta(\sigma),\ \Gamma, v : \sigma' \vdash_{\mathcal{L}} M : \tau \ \&\ \Gamma \vdash_{\mathcal{L}} V : \sigma$;

(iii) $\Gamma \vdash_{\mathcal{L}} M[W/\chi] : \tau \ \&\ W \in \mathsf{TVal} \Leftrightarrow \exists \sigma \in \mathit{CType},\ \Gamma, \chi : \sigma \vdash_{\mathcal{L}} M : \tau \ \&\ \Gamma \vdash_{\mathcal{L}} W : \sigma$.

Proof. (i) ($\Rightarrow$) If x does not occur in M we can choose $\sigma \equiv \omega$. Otherwise let σ be the intersection of all predicates of statements with subject N which occur in a given deduction of $\Gamma \vdash_{\mathcal{L}} M[N/x] : \tau$. The proof of $\Leftarrow$ is standard.

(ii) If v does not occur in M we can choose $\sigma \equiv \omega \to \omega$. Otherwise let ρ be the intersection of all predicates of statements with subject V which occur in a given deduction of $\Gamma \vdash_{\mathcal{L}} M[V/v] : \tau$. If $\rho \neq \omega$ we can choose $\sigma \equiv \rho$; otherwise $\sigma \equiv \omega \to \omega$.

(iii)($\Rightarrow$) If χ does not occur in M we can choose $\sigma \equiv \omega \to \omega$. Otherwise let ρ be the intersection of all predicates of statements with subject W which occur in a given deduction of $\Gamma \vdash_{\mathcal{L}} M[W/\chi] : \tau$. By Theorem 3.15(ii) there is $\rho' \in \Theta(\rho)$ such that $\Gamma \vdash_{\mathcal{L}} W : \rho'$. If $\rho' \neq \omega$ we can choose $\sigma \equiv \rho'$; otherwise $\sigma \equiv \omega \to \omega$. The proof of $\Leftarrow$ is standard. □

Notice that in Lemma 3.16(ii) the "$\Rightarrow$" cannot be replaced by "$\Leftrightarrow$". An easy proof of this uses the characterization of divergent terms by types which will be given in Corollary 5.6(ii). So we will prove it in Corollary 5.7(i).

As an immediate consequence of Theorem 3.15(ii) and Lemma 3.16(iii), the following rule ($\vee$E) is admissible:

$$(\vee\mathrm{E})\quad \frac{\forall \sigma' \in \Theta(\sigma) \quad \Gamma, \chi : \sigma' \vdash M : \rho \qquad \Gamma \vdash W : \sigma \qquad W \in \mathsf{TVal}}{\Gamma \vdash M[W/\chi] : \rho}\ .$$

Therefore, the restriction over the basis can be relaxed, allowing $\Gamma(v)$ to be any type different from ω. This would have the advantage of having a unique rule for abstraction, i.e., the standard one, avoiding ($\to \mathrm{I}_v$), which is a rule schema. Of course, rule ($\vee$E) should be added in this case. The reason why we choose the present less elegant version is that it greatly simplifies proofs.

3.3. The logic congruence relation. We now introduce the logical equivalence $\simeq^{\mathcal{L}}$; thereafter, we shall use the properties stated in Theorem 3.13 to establish the basic (in)equalities holding under this notion of equivalence. The invariance of types with respect to $\bowtie$ and to the reduction relation studied in section 2 will follow.

DEFINITION 3.17. *Let* $M, N \in \Lambda_{+\|}$; *then*

(i) $M \sqsubseteq^{\mathcal{L}} N \ \Leftrightarrow\ \forall \Gamma, \sigma,\ \Gamma \vdash_{\mathcal{L}} M : \sigma \Rightarrow \Gamma \vdash_{\mathcal{L}} N : \sigma$;

(ii) $\simeq^{\mathcal{L}} = \sqsubseteq^{\mathcal{L}} \cap \sqsupseteq^{\mathcal{L}}$.

As a first step in the study of the relation $\simeq^{\mathcal{L}}$ we fix some basic properties of it with respect to the various kinds of β-contraction present in our calculus. These can be easily proved using Lemma 3.16.

LEMMA 3.18.

(i) $(\lambda x.M)N \simeq_{\mathcal{L}} M[N/x]$;

(ii) $M[V/v] \sqsubseteq^{\mathcal{L}} (\lambda v.M)V$ *if* $V \in \mathsf{Val}$;

(iii) $(\lambda v.M)W \simeq^{\mathcal{L}} M[W/v]$ *if* $W \in \mathsf{TVal}$.

Proof. The most interesting case is the inclusion from left to right of (iii) when $\tau \neq \omega$.

$\Gamma \vdash_{\mathcal{L}} (\lambda v.M)W : \tau$
$\Rightarrow \quad \exists \sigma,\ \Gamma \vdash_{\mathcal{L}} \lambda v.M : \sigma \to \tau \ \&\ \Gamma \vdash_{\mathcal{L}} W : \sigma$ by Theorem 3.13(vi),
$\Rightarrow \quad \exists \sigma,\ \forall \sigma' \in \Theta(\sigma),\ \Gamma, v : \sigma' \vdash_{\mathcal{L}} M : \tau \ \&\ \Gamma \vdash_{\mathcal{L}} W : \sigma$ by Theorem 3.13(iv),
$\Rightarrow \quad \exists \sigma' \in CType,\ \Gamma, v : \sigma' \vdash_{\mathcal{L}} M : \tau \ \&\ \Gamma \vdash_{\mathcal{L}} W : \sigma'$ by Theorem 3.15(ii), since $W \in \mathsf{TVal}$,
$\Rightarrow \quad \Gamma \vdash_{\mathcal{L}} M[W/v] : \tau$ by Lemma 3.16(iii)($\Leftarrow$). □

Notice that the opposite of Lemma 3.18(ii) does not hold. This will be proved in Corollary 5.7(ii), since it follows immediately from point (i) of the same corollary.

The following three lemmas are easy consequences of Theorem 3.13. The second and third lemmas state that nondeterministic choice and parallel composition are the meet and the join, respectively. Moreover, they illustrate the behaviors of these operators with respect to application and abstraction.

LEMMA 3.19. *The relation $\simeq^{\mathcal{L}}$ is a congruence over $\Lambda_{+\|}$.*

LEMMA 3.20.

(i) $M + N \sqsubseteq^{\mathcal{L}} M, N$;
(ii) $L \sqsubseteq^{\mathcal{L}} M, N \Rightarrow L \sqsubseteq^{\mathcal{L}} M + N$;
(iii) $(M + N)L \simeq^{\mathcal{L}} ML + NL$;
(iv) $L(M + N) \sqsubseteq^{\mathcal{L}} LM + LN$;
(v) $(\lambda v.M)(N + L) \simeq^{\mathcal{L}} (\lambda v.M)N + (\lambda v.M)L$;
(vi) $\lambda\chi.(M + N) \sqsubseteq^{\mathcal{L}} \lambda\chi.M + \lambda\chi.N$.

Proof. All inclusions are immediate. The converse of (vi) does not hold. Indeed, let $\sigma \equiv (\rho \to \rho) \vee (\tau \to \omega^2 \to \omega)$ where $\rho \equiv \omega^3 \to \omega$ and $\tau \equiv (\omega \to \omega) \to \omega^2 \to \omega$. Then we have $\vdash_{\mathcal{L}} \mathbf{I} : \rho \to \rho$ and $\vdash_{\mathcal{L}} \Delta : \tau \to \omega^2 \to \omega$, which imply $\vdash_{\mathcal{L}} \mathbf{I} + \Delta : \sigma$, but $\not\vdash_{\mathcal{L}} \lambda x.(x + xx) : \sigma$. In fact, by Theorem 3.13(iii) and (vii), if we could derive $\lambda x.(x+xx) : \sigma$, then we would also have $x : \mu \vdash_{\mathcal{L}} x : \nu$ and $x : \mu \vdash_{\mathcal{L}} xx : \nu$ for some μ, ν such that $\mu \to \nu \leq \sigma$. This implies either $\mu \to \nu \leq \rho \to \rho$ or $\mu \to \nu \leq \tau \to \omega^2 \to \omega$ by Definition 3.6 and Proposition 3.8. But it is easy to verify, using Theorem 3.13(i) and (vi), that $x : \rho \not\vdash_{\mathcal{L}} xx : \rho$ and $x : \tau \not\vdash_{\mathcal{L}} x : \omega^2 \to \omega$. □

LEMMA 3.21.

(i) $M, N \sqsubseteq^{\mathcal{L}} M\|N$;
(ii) $M, N \sqsubseteq^{\mathcal{L}} L \Rightarrow M\|N \sqsubseteq^{\mathcal{L}} L$;
(iii) $(M\|N)L \simeq^{\mathcal{L}} ML\|NL$;
(iv) $LM\|LN \sqsubseteq^{\mathcal{L}} L(M\|N)$;
(v) $\lambda\chi.(M\|N) \simeq^{\mathcal{L}} \lambda\chi.M\|\lambda\chi.N$;
(vi) $(M + N)\|L \simeq^{\mathcal{L}} M\|L + N\|L$.

The inequalities of Lemma 3.20(iv) and 3.21(iv) are proper, and this can be proved using the structural properties of deductions (Theorem 3.13). But an easier proof will be given in Corollary 5.7(iii) and (iv) using Corollary 5.6(ii).

The following theorem provides the first evidence of the matching between operational and logic semantics.

THEOREM 3.22 (type invariance).

(i) $\Gamma \vdash_{\mathcal{L}} M : \sigma \ \&\ M \bowtie N \Rightarrow \Gamma \vdash_{\mathcal{L}} N : \sigma$;
(ii) $\Gamma \vdash_{\mathcal{L}} M : \sigma \ \&\ M \longrightarrow^* N \Rightarrow \Gamma \vdash_{\mathcal{L}} N : \sigma$.

Proof. (i) is an easy consequence of Lemmas 3.18, 3.19, 3.20, and 3.21.

(ii) If $M \longrightarrow^* N$ then for some $b \in bar(M)$ it is the case that $N \in b$. By Lemma 2.9 (ii) $M \rhd \sum\{M' \mid M' \in b\}$. Being $\rhd \subseteq \bowtie$, by part (i) of the present theorem and Theorem 3.13(vii), we have $\Gamma \vdash_{\mathcal{L}} M' : \sigma$ for all $M' \in b$, from which the thesis follows. □

The subject expansion property fails for $\xrightarrow{*}$. For example, $\vdash_{\mathcal{L}} \mathbf{I} : \omega \to \omega$ but, as we shall be able to derive from Corollary 5.6(ii), $\not\vdash_{\mathcal{L}} \mathbf{I} + \Omega : \omega \to \omega$.

The main result of the present section is that convergence implies typability by $\omega \to \omega$. We will see in section 5 that the converse is also true. Therefore, this type will completely characterize terms whose meaning is to eventually be a function, even if not a unique one.

THEOREM 3.23. *Let M be a closed term:*

$$M\Downarrow \;\Rightarrow\; \vdash_{\mathcal{L}} M : \omega \to \omega.$$

Proof.

$$\begin{array}{lll} M\Downarrow & \Rightarrow & \exists V_1, \ldots, V_n \in \mathsf{Val},\ M \rhd \sum_{i=1}^{n} V_i \quad \text{by Corollary 2.10(i),} \\ & \Rightarrow & \vdash_{\mathcal{L}} \sum_{i=1}^{n} V_i : \omega \to \omega \quad \text{by Theorem 3.15(i) and rule } (+\mathrm{I}), \\ & \Rightarrow & \vdash_{\mathcal{L}} M : \omega \to \omega \quad \text{by Theorem 3.22(i).} \end{array}$$

□

4. Models and completeness. If we want to devise a domain equation for our concurrent λ-calculus, it is natural to start from the equations in the literature for similar languages.

Abramsky, in [4], interprets the lazy λ-calculus by means of a Scott domain D solving the equation

$$D = [D \to D]_{\perp},$$

where $[D \to D]$ is the space of continuous functions and $(\cdot)_{\perp}$ is the lifting operator.

Boudol in [19] gives the semantics of the lazy, call-by-name and call-by-value λ-calculus enriched with a parallel operator using the same equation, but in a different category. It is easy to see from the asynchronous reduction rules of Boudol's parallel operator (shown on p. 1379) that in a "may" perspective $\|$ can be interpreted using the lower powerdomain. Boudol recalls that each prime algebraic lattice D is isomorphic to the lower powerdomain of the posets of the compact coprime elements of D. Therefore it suffices to find a solution of Abramsky's equation in this category to have a domain suitable for Boudol's language. Notice that Boudol interprets $M\|N$ as the join of the interpretations of M and N.

The reduction rules of the present parallel operator differ from those given in [19]. Really, our $\|$ is synchronous. But we are in a different situation: we consider "must" convergence instead of "may" convergence. Therefore, our parallel operator behaves exactly like Boudol's from the viewpoint of convergence. In fact, both operators converge whenever one of the two arguments does. This is clear when we think of the correspondence between asynchronicity in a "may" perspective and synchronicity in a "must" perspective. So we could have used Abramsky's domain equation again, if we would not have to also interpret the nondeterministic choice.

The reduction rules of $+$ in a "must" perspective clearly suggest the upper powerdomain for its interpretation. The whole discussion leads to the following domain equation:

$$D = \mathcal{P}^{\sharp}([D \to D]_{\perp}),$$

where $\mathcal{P}^{\sharp}$ is the upper powerdomain functor, in the category of prime algebraic lattices.

It is well known that each prime algebraic lattice can be described by an information system [42] and also by means of intersection types [22]. Really, we have

developed in previous sections a system of intersection and union types; we will use this system now to build a model, which is actually the initial solution of our domain equation.

Because of rules $(\omega), (\leq)$, and $(\wedge\ \mathrm{I})$, the set of types assigned in $\mathcal{L}$ to any term is a filter over $Type$: let $\mathcal{F}$ be the set of all filters. $\mathcal{F}$ is a distributive lattice under subset ordering (distributivity comes as a consequence of the distributivity of $Type$ itself), with intersection as meet and

$$F \bar{\cup} F' = \uparrow \{\sigma \wedge \tau \mid \sigma \in F, \tau \in F'\}$$

as join ($\uparrow$ stands as usual for upper closure). The bottom and the top of this lattice are, respectively, $\uparrow \omega$ and $Type$, where in general $\uparrow \sigma$ is the principal filter generated by σ. The compact elements are the principal filters. Moreover, this lattice is prime algebraic, since each filter is the join of the compact coprime filters it dominates. Notice that a filter $F \in \mathcal{F}$ is compact *coprime* if and only if it is a principal filter generated by a *meet irreducible* type.[2] We refer to [7] for the whole proof that $\mathcal{F}$ is the initial solution of our domain equation.

Among filters assigned as meanings of terms, Theorem 3.15(ii) indicates that prime filters are the interpretations of terms that are total values. We recall that a filter $F \in \mathcal{F}$ is *prime* if and only if for all σ and τ

$$\sigma \vee \tau \in F \Rightarrow \sigma \in F \text{ or } \tau \in F.$$

We write $\mathcal{F}_P$ to denote the set of prime filters.

In any distributive lattice D the set $\mathrm{Pr}(D)$ of *prime elements* is defined as follows:

$$d \in \mathrm{Pr}(D) \ \Leftrightarrow\ \forall x, y \in D,\ x \sqcap y \sqsubseteq d \Rightarrow x \sqsubseteq d \text{ or } y \sqsubseteq d.$$

We write $\mathrm{Pr}(x) = \uparrow x \cap \mathrm{Pr}(D)$ for $x \in D$. Let us define, for any filter F, the set

$$\mathrm{Pr}(F) = \{P \in \mathcal{F}_P \mid F \subseteq P\},$$

which is called the *prime decomposition* of F. It is straightforward to see that $\mathrm{Pr}(\mathcal{F}) = \mathcal{F}_P$ and consequently that the previous definition of $\mathrm{Pr}(F)$ is consistent with the notation $\mathrm{Pr}(x)$.

From Priestley's theorem we know that the structure of a distributive lattice is recoverable from its prime filters (or dually from its prime ideals). The following fact is at the root of this result (see, e.g., [23, Theorem 10.3]):

(DPI) Let D be a distributive lattice, F a filter, and I an ideal in D such that $F \cap I = \emptyset$. Then there exists a prime filter P and a prime ideal J (actually, J is the complement of P in D) such that $F \subseteq P$, $I \subseteq J$, and $P \cap J = \emptyset$.

The principle (DPI) implies that each filter is completely determined by its prime decomposition.

LEMMA 4.1. $\forall F \in \mathcal{F},\ F = \bigcap_{P \in \mathrm{Pr}(F)} P$.

Proof. The left to right inclusion is immediate. To see the inverse inclusion let us suppose in contradiction that there exists some $\sigma \in \bigcap_{P \in \mathrm{Pr}(F)} P$ such that $\sigma \notin F$. This implies that $\downarrow \sigma \cap F = \emptyset$, where $\downarrow \sigma$ is the principal ideal generated by σ; it follows by (DPI) that for some $P \in \mathcal{F}_P$ we have $F \subseteq P$ and $\downarrow \sigma \cap P = \emptyset$. □

The last lemma is an instance of a more general fact: let D be a lattice; then $X \subseteq D$ is *order generating* if and only if for all $x \in D$, $x = \sqcap(\uparrow x \cap X)$ (see [30, Ch. 1,

[2] A type σ is *meet irreducible* or *prime* if and only if $\tau \wedge \rho \leq \sigma \Rightarrow \tau \leq \sigma$ or $\rho \leq \sigma$ for any τ, ρ.

Definition 3.8]). If D is continuous (i.e., complete and each element is the sup of its way below elements) then it is distributive if and only if $\mathrm{Pr}(D)$ is order generating (see [30, Ch. 1, Theorem 3.14]). But $\mathcal{F}$ is a distributive lattice which is prime algebraic, so it is a fortiori continuous. Therefore, $\mathcal{F}_P$ is order generating.

To interpret functional application we turn $\mathcal{F}$ into an applicative structure as follows:

$$F \cdot F' = \{\tau \mid \exists \sigma \in F',\ \sigma \to \tau \in F\} \ \cup \{\uparrow \omega\}.$$

Observe that the definition of application is slightly different from that given in [16]. Indeed, we have to add explicitly the principal filter of ω, since in our setting $\omega \neq \omega \to \omega$; otherwise $\uparrow\omega \cdot \uparrow\omega$ would be the empty set.

LEMMA 4.2. *The operation of application over $\mathcal{F}$ is monotonic in both its arguments*; *moreover,*

$$(F \cap F') \cdot G \supseteq (F \cdot G) \cap (F' \cdot G) \quad \textit{and} \quad (F \cup F') \cdot G \subseteq (F \cdot G) \cup (F' \cdot G)$$

for all $F, F', G \in \mathcal{F}$.

The proof is straightforward. Just note that these inclusions are actually equalities, since the opposite inclusions follow from the monotonicity of the application.

The properties of $\mathcal{F}$ which have been seen so far suggest the following definition.

DEFINITION 4.3. *A* premodel *of $\Lambda_{+\|}$ is a structure $\mathcal{D} = \langle D, \sqsubseteq, \cdot, \sqcap, \sqcup \rangle$ where $\langle D, \sqsubseteq \rangle$ is a distributive continuous lattice and $\cdot$ is a monotonic binary operation on D such that, for all $d, d', e \in D$,*

(i) $(d \sqcap d') \cdot e \sqsupseteq (d \cdot e) \sqcap (d' \cdot e)$,

(ii) $(d \sqcup d') \cdot e \sqsubseteq (d \cdot e) \sqcup (d' \cdot e)$.

Total values are associated by system $\mathcal{L}$ to prime filters different from $\uparrow\omega$. A call-by-value variable is a total value; hence a correct notion of environment for $\mathcal{F}$ is a mapping $\eta : \mathsf{Vn} \cup \mathsf{Vv} \to \mathcal{F}$ such that $\eta(\mathsf{Vv}) \subseteq \mathcal{F}_P - \{\uparrow\omega\}$. In general, given a premodel $\mathcal{D}$, if $\mathcal{P} = \mathrm{Pr}(D) - \{\bot\}$, we define Env_D as the set of mappings $\eta : \mathsf{Vn} \cup \mathsf{Vv} \to D$ such that $\eta(\mathsf{Vv}) \subseteq \mathcal{P}$.

Now, for any environment $\eta \in Env_{\mathcal{F}}$ and for any basis Γ, we define

$$\Gamma \models \eta \ \Leftrightarrow\ \forall \chi \in \mathsf{Vn} \cup \mathsf{Vv},\ \Gamma(\chi) \in \eta(\chi).$$

We are now set to define the map $[\![\cdot]\!]^{\mathcal{F}} : \Lambda_{+\|} \to Env_{\mathcal{F}} \to \mathcal{F}$ as follows:

$$[\![M]\!]^{\mathcal{F}}_{\eta} = \{\sigma \mid \exists \Gamma,\ \Gamma \models \eta \ \&\ \Gamma \vdash_{\mathcal{L}} M : \sigma\}.$$

This definition is consistent with the logical inclusion, which is equivalent to subset inclusion of interpretations.

PROPOSITION 4.4. *For all $M, N \in \Lambda_{+\|}$,*

$$M \sqsubseteq^{\mathcal{L}} N \ \Leftrightarrow\ \forall \eta,\ [\![M]\!]^{\mathcal{F}}_{\eta} \subseteq [\![N]\!]^{\mathcal{F}}_{\eta}\ .$$

Proof. ($\Rightarrow$) The proof is immediate. ($\Leftarrow$) Let us define, for any basis Γ, $\eta_\Gamma(\chi) = \uparrow \Gamma(\chi)$ for all variables χ; then $\eta_\Gamma \in Env_{\mathcal{F}}$ since $\Gamma(v)$ is coprime for all call-by-value variables v, and hence $\uparrow\Gamma(v)$ is a prime filter. Now $\Gamma \models \eta_\Gamma$ so that $\Gamma \vdash_{\mathcal{L}} M : \sigma$ implies $\sigma \in [\![M]\!]^{\mathcal{F}}_{\eta_\Gamma}$. By hypothesis, $\sigma \in [\![N]\!]^{\mathcal{F}}_{\eta_\Gamma}$, hence $\Gamma' \vdash_{\mathcal{L}} N : \sigma$ for some Γ' such that $\Gamma' \models \eta_\Gamma$. We conclude that $\Gamma \vdash_{\mathcal{L}} N : \sigma$ since $\Gamma' \models \eta_\Gamma$ implies $\Gamma \leq \Gamma'$. □

COROLLARY 4.5. *For all* $M, N \in \Lambda_{+\|}$ *and* $\eta \in Env_{\mathcal{F}}$

$$[\![M+N]\!]^{\mathcal{F}}_{\eta} = [\![M]\!]^{\mathcal{F}}_{\eta} \cap [\![N]\!]^{\mathcal{F}}_{\eta} \quad \textit{and} \quad [\![M\|N]\!]^{\mathcal{F}}_{\eta} = [\![M]\!]^{\mathcal{F}}_{\eta} \cup [\![N]\!]^{\mathcal{F}}_{\eta} \ .$$

Proof. The proof is immediate from Proposition 4.4 and from Lemmas 3.20(i), (ii) and 3.21(i), (ii). □

Elaborating on the definition of λ-model, and also on the notion of λ-lattice proposed in [25], we fix the following.

DEFINITION 4.6. *The structure* $\langle \mathcal{D}, [\![\cdot]\!]^{\mathcal{D}} \rangle$ *is a* model *if* $\mathcal{D} = \langle D, \sqsubseteq, \cdot, \sqcap, \sqcup \rangle$ *is a premodel and* $[\![\cdot]\!]^{\mathcal{D}} : \Lambda_{+\|} \to Env_D \to D$ *satisfies the following conditions:*

(i) $[\![\chi]\!]^{\mathcal{D}}_{\eta} = \eta(\chi)$;
(ii) $[\![MN]\!]^{\mathcal{D}}_{\eta} = [\![M]\!]^{\mathcal{D}}_{\eta} \cdot [\![N]\!]^{\mathcal{D}}_{\eta}$;
(iii) $[\![\lambda x.M]\!]^{\mathcal{D}}_{\eta} \cdot d = [\![M]\!]^{\mathcal{D}}_{\eta[x \mapsto d]}$;
(iv) $[\![\lambda v.M]\!]^{\mathcal{D}}_{\eta} \cdot d = \begin{cases} \bot & \textit{if } d = \bot, \\ \sqcap_{e \in \mathrm{Pr}(d)} [\![M]\!]^{\mathcal{D}}_{\eta[v \mapsto e]} & \textit{otherwise}; \end{cases}$
(v) $(\forall \chi \in \mathrm{FV}(M),\ \eta(\chi) = \eta'(\chi)) \Rightarrow [\![M]\!]^{\mathcal{D}}_{\eta} = [\![M]\!]^{\mathcal{D}}_{\eta'}$;
(vi) $[\![\lambda \chi.M]\!]^{\mathcal{D}}_{\eta} = [\![\lambda \chi'.M[\chi'/\chi]]\!]^{\mathcal{D}}_{\eta}$ *if* $\chi' \notin \mathrm{FV}(M)$ *and either* $\chi, \chi' \in \mathsf{Vn}$ *or* $\chi, \chi' \in \mathsf{Vv}$;
(vii) $(\forall d \in D,\ [\![M]\!]^{\mathcal{D}}_{\eta[x \mapsto d]} = [\![N]\!]^{\mathcal{D}}_{\eta[x \mapsto d]}) \Rightarrow [\![\lambda x.M]\!]^{\mathcal{D}}_{\eta} = [\![\lambda x.N]\!]^{\mathcal{D}}_{\eta}$;
(viii) $(\forall e \in \mathcal{P},\ [\![M]\!]^{\mathcal{D}}_{\eta[v \mapsto e]} = [\![N]\!]^{\mathcal{D}}_{\eta[v \mapsto e]}) \Rightarrow [\![\lambda v.M]\!]^{\mathcal{D}}_{\eta} = [\![\lambda v.N]\!]^{\mathcal{D}}_{\eta}$;
(ix) $[\![M+N]\!]^{\mathcal{D}}_{\eta} = [\![M]\!]^{\mathcal{D}}_{\eta} \sqcap [\![N]\!]^{\mathcal{D}}_{\eta}$;
(x) $[\![M\|N]\!]^{\mathcal{D}}_{\eta} = [\![M]\!]^{\mathcal{D}}_{\eta} \sqcup [\![N]\!]^{\mathcal{D}}_{\eta}$;
(xi) $W \in \mathsf{TVal} \Rightarrow [\![W]\!]^{\mathcal{D}}_{\eta} \in \mathcal{P}$,

where $\mathcal{P} = \mathrm{Pr}(D) - \{\bot\}$.

With respect to the classical definition of (syntactical) λ-models, the novelties are in clauses (iv) and (viii)–(x). Clause (xi) reflects the intended meaning of total values, which essentially are not sums. Clause (viii) takes into account that by definition, $\eta(\mathsf{Vv}) \subseteq \mathcal{P}$. The last two clauses are suggested by Corollary 4.5. Clause (iv) is more demanding; indeed, from Corollary 3.14 and Proposition 4.4 we can argue that a call-by-value abstraction defines a co-additive function, but this does not suffice to show that it is completely co-additive (i.e., preserving arbitrary meets). This is, however, true in the filter model, and finely fits into the fact that prime elements are order generating in continuous distributive lattices. To show this, or equivalently that the premodel $\mathcal{F}$ can be turned into a model using $[\![\cdot]\!]^{\mathcal{F}}$, we need a couple of lemmas.

LEMMA 4.7. *For all* $M, N \in \Lambda_{+\|}$ *and* $\eta \in Env_{\mathcal{F}}$,

$$[\![MN]\!]^{\mathcal{F}}_{\eta} = [\![M]\!]^{\mathcal{F}}_{\eta} \cdot [\![N]\!]^{\mathcal{F}}_{\eta}.$$

Proof. To prove $[\![MN]\!]^{\mathcal{F}}_{\eta} \subseteq [\![M]\!]^{\mathcal{F}}_{\eta} \cdot [\![N]\!]^{\mathcal{F}}_{\eta}$, let $\sigma \in [\![MN]\!]^{\mathcal{F}}_{\eta}$ and $\sigma \neq \omega$; then for some basis Γ we have $\Gamma \models \eta$ and $\Gamma \vdash_{\mathcal{L}} MN : \sigma$. By Theorem 3.13(vi), there exists some τ such that

$$\Gamma \vdash_{\mathcal{L}} M : \tau \to \sigma \ \&\ \Gamma \vdash_{\mathcal{L}} N : \tau.$$

It follows that $\tau \to \sigma \in [\![M]\!]^{\mathcal{F}}_{\eta}$ and $\tau \in [\![N]\!]^{\mathcal{F}}_{\eta}$, so the thesis follows.

To see that $[\![M]\!]^{\mathcal{F}}_\eta \cdot [\![N]\!]^{\mathcal{F}}_\eta \subseteq [\![MN]\!]^{\mathcal{F}}_\eta$ we reason as in [16]; namely, if for some $\tau \in [\![N]\!]^{\mathcal{F}}_\eta$ it is the case that $\tau \to \sigma \in [\![M]\!]^{\mathcal{F}}_\eta$, then there are two bases, Γ_0, Γ_1, such that $\Gamma_i \models \eta$ for $i = 0, 1$, and

$$\Gamma_0 \vdash_{\mathcal{L}} N : \tau \ \&\ \Gamma_1 \vdash_{\mathcal{L}} M : \tau \to \sigma.$$

Now, taking Γ_2 such that $\Gamma_2(\chi) = \Gamma_0(\chi) \wedge \Gamma_1(\chi)$ for all χ, it is easy to see that

$$\Gamma_2 \models \eta \ \&\ \Gamma_2 \vdash_{\mathcal{L}} N : \tau \ \&\ \Gamma_2 \vdash_{\mathcal{L}} M : \tau \to \sigma,$$

from which we get $\Gamma_2 \vdash_{\mathcal{L}} MN : \sigma$; that is, $\sigma \in [\![MN]\!]^{\mathcal{F}}_\eta$. □

LEMMA 4.8. *Let $\Sigma = \{\sigma_i\}_{i\in\mathbb{N}}$ be a chain such that for all i, $\sigma_i \leq \sigma_{i+1}$, and let $F \in \mathcal{F}$. Then*

$$\forall P \in \mathrm{Pr}(F),\ P \cap \Sigma \neq \emptyset \Rightarrow F \cap \Sigma \neq \emptyset.$$

Proof. Let I be the downward closure of Σ; then

$$\begin{array}{rcl} \rho_0, \rho_1 \in I & \Rightarrow & \exists \sigma_i, \sigma_j \in \Sigma,\ \rho_0 \leq \sigma_i \ \&\ \rho_1 \leq \sigma_j \\ & \Rightarrow & \exists \sigma_i, \sigma_j \in \Sigma,\ \rho_0 \vee \rho_1 \leq \sigma_i \vee \sigma_j = \sigma_{\max\{i,j\}} \in \Sigma \\ & \Rightarrow & \rho_0 \vee \rho_1 \in I, \end{array}$$

so I is an ideal. If $F \cap \Sigma = \emptyset$, then $F \cap I = \emptyset$, F being an upward closed set. By (DPI) there exists $P \in \mathrm{Pr}(F)$ such that $P \cap I = \emptyset$ and consequently $P \cap \Sigma = \emptyset$, so that the thesis follows by contraposition. □

THEOREM 4.9. *The structure $\langle \langle \mathcal{F}, \subseteq, \cdot, \cap, \bar{\cup} \rangle, [\![\cdot]\!]^{\mathcal{F}} \rangle$ is a model.*

Proof. Because of Proposition 4.4 and Lemmas 3.18, 3.20, 3.21, and 4.7 the only relevant remaining point is to show that $\mathcal{F}$ satisfies clause (iv) of Definition 4.6. Recall that the bottom in $\mathcal{F}$ is $\uparrow\omega$; then this amounts to showing that

$$[\![\lambda v.M]\!]^{\mathcal{F}}_\eta \cdot F = \begin{cases} \uparrow\omega & \text{if } F \equiv \uparrow\omega, \\ \bigcap_{P\in\mathrm{Pr}(F)} [\![M]\!]^{\mathcal{F}}_{\eta[v\mapsto P]} & \text{otherwise.} \end{cases}$$

Now, if $F \equiv \uparrow\omega$, then

$$\begin{array}{rcll} \sigma \in [\![\lambda v.M]\!]^{\mathcal{F}}_\eta \cdot \uparrow\omega & \Leftrightarrow & \omega \to \sigma \in [\![\lambda v.M]\!]^{\mathcal{F}}_\eta & \\ & \Leftrightarrow & \exists \Gamma \models \eta,\ \Gamma \vdash_{\mathcal{L}} \lambda v.M : \omega \to \sigma & \\ & \Rightarrow & \sigma = \omega & \text{by Theorem 3.13(v).} \end{array}$$

If $F \not\equiv \uparrow\omega$, let us suppose that $\sigma \neq \omega$ (otherwise the thesis is trivial). Then

$$\begin{array}{rcl} \sigma \in [\![\lambda v.M]\!]^{\mathcal{F}}_\eta \cdot F & \Leftrightarrow & \exists \tau \in F,\ \tau \to \sigma \in [\![\lambda v.M]\!]^{\mathcal{F}}_\eta \\ & \Leftrightarrow & \exists \tau \in F,\ \exists \Gamma \models \eta,\ \Gamma \vdash_{\mathcal{L}} \lambda v.M : \tau \to \sigma \\ & \Leftrightarrow & \exists \tau \in F,\ \exists \Gamma \models \eta,\ \forall \tau' \in \Theta(\tau),\ \Gamma, v : \tau' \vdash_{\mathcal{L}} M : \sigma \\ & & \text{by Theorem 3.13(iv), since } \sigma \neq \omega \Rightarrow \tau \neq \omega \\ & & \text{by Theorem 3.13(v).} \end{array}$$

Now $F = \bigcap_{P\in\mathrm{Pr}(F)} P$ implies that, for all $P \in \mathrm{Pr}(F)$, $\tau \in P$, and hence $\tau' \in P$ for some $\tau' \in \Theta(\tau)$ by definition of prime filter (notice that τ' depends on P). This implies $\Gamma, v : \tau' \models \eta[v \mapsto P]$ for some $\tau' \in \Theta(\tau)$, so that $\sigma \in [\![M]\!]^{\mathcal{F}}_{\eta[v\mapsto P]}$. It follows that

$$[\![\lambda v.M]\!]^{\mathcal{F}}_\eta \cdot F \subseteq \bigcap_{P\in\mathrm{Pr}(F)} [\![M]\!]^{\mathcal{F}}_{\eta[v\mapsto P]}.$$

To see the opposite inclusion, let $G = [\![\lambda v.M]\!]^{\mathcal{F}}_{\eta}$; we first show that, if $P \in \mathcal{F}_P$ and $P \neq \uparrow \omega$, then $G \cdot P = [\![M]\!]_{\eta[v \mapsto P]}$. Indeed, let $w \in \mathsf{Vv} - FV(M)$ and $\eta \in Env_{\mathcal{F}}$ be such that $\eta(w) = P$; then, by Lemmas 4.7 and 3.18(iii), we have

$$G \cdot P = [\![\lambda v.M]\!]^{\mathcal{F}}_{\eta} \cdot [\![w]\!]^{\mathcal{F}}_{\eta} = [\![(\lambda v.M)w]\!]^{\mathcal{F}}_{\eta} = [\![M[w/v]]\!]^{\mathcal{F}}_{\eta} = [\![M]\!]^{\mathcal{F}}_{\eta[v \mapsto P]}.$$

From $F \neq \uparrow \omega$ it follows that $\uparrow \omega \notin \mathrm{Pr}(F)$, so that the equality above implies

$$\bigcap_{P \in \mathrm{Pr}(F)} G \cdot P \;=\; \bigcap_{P \in \mathrm{Pr}(F)} [\![M]\!]^{\mathcal{F}}_{\eta[v \mapsto P]}.$$

Suppose that $\sigma \in \bigcap_{P \in \mathrm{Pr}(F)} G \cdot P$. Let us define

$$\Pi = \{\pi \in Type \mid \exists P \in \mathrm{Pr}(F),\ \pi \in P \ \&\ \pi \to \sigma \in G\}.$$

This set is nonempty by hypothesis and it is countable, being a subset of the denumerable set $Type$. Let us suppose that an enumeration of Π has been fixed; then we put

$$\Sigma = \{\tau_n\}_{n \in \mathbb{N}} \quad \text{where} \quad \tau_n = \bigvee_{i=0}^{n} \pi_i.$$

By construction, Σ is a chain such that, $\forall P \in \mathrm{Pr}(F),\ P \cap \Sigma \neq \emptyset$, so that by Lemma 4.8, there exists an m such that $\tau_m \in F$. Since G is the interpretation of a call-by-value abstraction, a simple induction using Corollary 3.14 shows that $\tau_n \to \sigma \in G$ for all n: we conclude that $\sigma \in G \cdot F$. □

So far, terms have been interpreted as collections of their properties, namely, of their types. We now provide an interpretation of types as subsets of the domains of models. The domains have an order whose meaning is "to be better behaved" and more defined. We do not want more defined objects to have fewer properties than their minors, hence the subsets interpreting types have to be upward closed; more precisely, they will be filters of the domain itself.

DEFINITION 4.10. *Let $\mathcal{D} = \langle D, \sqsubseteq, \cdot, \sqcap, \sqcup \rangle$ be a premodel, and define for $X, Y \subseteq D$:*

$$X \Rightarrow Y = \{d \in D \mid d \neq \bot \ \&\ \forall e \in X,\ d \cdot e \in Y\},$$
$$X \bar{\cup} Y = \uparrow \{d \sqcap d' \mid d \in X, d' \in Y\},$$

where we overload $\bar{\cup}$. Then a type structure *over $\mathcal{D}$ is a sublattice $\mathcal{T}$ of the lattice of filters over $\mathcal{D}$, such that $D \in \mathcal{T}$ and $\mathcal{T}$ is closed under $\Rightarrow, \cap$, and $\bar{\cup}$.*
The map $[\![\cdot]\!]^{\mathcal{T}} : Type \to \mathcal{T}$ is inductively defined as follows:

(i) $[\![\omega]\!]^{\mathcal{T}} = D$;
(ii) $[\![\sigma \to \tau]\!]^{\mathcal{T}} = [\![\sigma]\!]^{\mathcal{T}} \Rightarrow [\![\tau]\!]^{\mathcal{T}}$;
(iii) $[\![\sigma \wedge \tau]\!]^{\mathcal{T}} = [\![\sigma]\!]^{\mathcal{T}} \cap [\![\tau]\!]^{\mathcal{T}}$;
(iv) $[\![\sigma \vee \tau]\!]^{\mathcal{T}} = [\![\sigma]\!]^{\mathcal{T}} \bar{\cup} [\![\tau]\!]^{\mathcal{T}}$.

Finally, given a model $\langle \mathcal{D}, [\![\cdot]\!]^{\mathcal{D}} \rangle$, we define

(v) $\Gamma \models_{\mathcal{D}} M : \sigma \ \Leftrightarrow\ (\forall \eta \in Env_D,\ \Gamma \models \eta \Rightarrow [\![M]\!]^{\mathcal{D}}_{\eta} \in [\![\sigma]\!]^{\mathcal{T}})$;
(vi) $\Gamma \models M : \sigma \ \Leftrightarrow\ \forall \mathcal{D},\ \Gamma \models_{\mathcal{D}} M : \sigma$.

In the definition of $X \Rightarrow Y$ we make the condition $d \neq \bot$ since we are modeling a lazy calculus; this means that the bottom cannot be interpreted as a function. Consequently we exclude $\bot$ from $X \Rightarrow Y$, whose intended meaning is the set of

representatives of functions which, when restricted to X, have ranges included in Y. Observe that $X \Rightarrow Y$ is a filter in D if both X and Y are.

We end this section by stating and proving a completeness theorem.

THEOREM 4.11 (completeness).

$$\Gamma \vdash_{\mathcal{L}} M : \sigma \;\Leftrightarrow\; \Gamma \models M : \sigma.$$

Proof. ($\Rightarrow$) The proof is by induction on derivations in $\mathcal{L}$.

($\Leftarrow$) First define $X_\sigma = \{F \in \mathcal{F} \mid \sigma \in F\}$ as usual, so that it is easily checked that $\mathcal{T} = \{X_\sigma\}_{\sigma \in Type}$ is a type structure over $\mathcal{F}$. More precisely, $X_\omega = \mathcal{F}$, $X_{\sigma\to\tau} = X_\sigma \Rightarrow X_\tau$, $X_{\sigma\wedge\tau} = X_\sigma \cap X_\tau$, and $X_{\sigma\vee\tau} = X_\sigma \bar{\cup} X_\tau$. This implies, by a simple induction on types, that $[\![\sigma]\!]^{\mathcal{T}} = X_\sigma$. Suppose $\Gamma \models M : \sigma$; hence, in particular, $\Gamma \models_{\mathcal{F}} M : \sigma$, since by Theorem 4.9, $\mathcal{F}$ is a model. Put $\eta_\Gamma(\chi) = \uparrow \Gamma(\chi)$. Now $\Gamma \models \eta_\Gamma$ implies $[\![M]\!]^{\mathcal{F}}_{\eta_\Gamma} \in X_\sigma$, that is, $\sigma \in [\![M]\!]^{\mathcal{F}}_{\eta_\Gamma}$. It follows that, for some Γ' such that $\Gamma' \models \eta_\Gamma$, we have $\Gamma' \vdash_{\mathcal{L}} M : \sigma$. We conclude that $\Gamma \vdash_{\mathcal{L}} M : \sigma$ since $\Gamma' \models \eta_\Gamma$ implies $\Gamma \leq \Gamma'$. □

5. Full abstraction. In this section we will prove that the filter model exactly mirrors the operational semantics; i.e., it is fully abstract. This means that

- the filter model is adequate; that is, it does not equate operationally distinct programs

$$M \sqsubseteq^{\mathcal{L}} N \Rightarrow M \sqsubseteq^{\mathcal{O}} N\ ;$$

- the filter model reflects the operational distinctions

$$M \sqsubseteq^{\mathcal{O}} N \Rightarrow M \sqsubseteq^{\mathcal{L}} N.$$

The key property on which the proof of full abstraction relies is that any compact element of $\mathcal{F}$, which is of the shape $\uparrow\sigma$, is λ-definable, since for all types σ there exists a characteristic (closed) term R_σ such that

$$\vdash_{\mathcal{L}} \mathsf{R}_\sigma : \tau \quad\Leftrightarrow\quad \sigma \leq \tau;$$

that is, $[\![\mathsf{R}_\sigma]\!]^{\mathcal{F}} = \uparrow\sigma$.

Such terms are constructed inductively together with test terms. To each type σ we associate a test term T_σ such that for all closed terms M:

$$\mathsf{T}_\sigma M{\Downarrow} \;\Leftrightarrow\; \vdash_{\mathcal{L}} M : \sigma.$$

The definition of characteristic and test terms finely reflects the duality between $\|$ and $+$, as well as their correspondence with $\wedge$ and $\vee$, respectively.

A further step in the full abstraction proof consists in giving a "realizability interpretation" of types as sets of closed terms. This is sound since each type is inhabited by some closed term (at least its characteristic term).

The main result we obtain is the converse of Theorem 3.23, namely,

$$(*) \qquad \vdash_{\mathcal{L}} M : \omega \to \omega \Rightarrow M{\Downarrow}$$

for all closed terms M.

The full abstraction theorem then follows. Indeed, adequacy is a consequence of $(*)$ and of the fact that $\simeq^{\mathcal{L}}$ is a congruence. For the converse it suffices to observe that test terms discriminate internally, that is, with respect to the convergence predicate, terms having different interpretations in the filter model.

5.1. Characteristic terms. We define two families of terms $\{\mathsf{R}_\sigma\}_{\sigma\in Type}$ and $\{\mathsf{T}_\sigma\}_{\sigma\in Type}$ starting from Ω . We can safely replace Ω by any unsolvable term of degree 0.

DEFINITION 5.1. *The* characteristic terms R_σ *and the* test terms T_σ *are defined by simultaneous induction on* σ*:*

$$\begin{array}{ll}
\mathsf{R}_\omega \equiv \Omega, & \mathsf{T}_\omega \equiv \lambda xy.y,\\
\mathsf{R}_{\sigma\to\tau} \equiv \lambda x.(\mathsf{T}_\sigma x)\,\mathsf{R}_\tau, & \mathsf{T}_{\sigma\to\tau} \equiv \lambda v.\mathsf{T}_\tau(v\,\mathsf{R}_\sigma),\\
\mathsf{R}_{\sigma\wedge\tau} \equiv \mathsf{R}_\sigma \| \mathsf{R}_\tau, & \mathsf{T}_{\sigma\wedge\tau} \equiv \lambda x.(\mathsf{T}_\sigma\, x + \mathsf{T}_\tau\, x),\\
\mathsf{R}_{\sigma\vee\tau} \equiv \mathsf{R}_\sigma + \mathsf{R}_\tau, & \mathsf{T}_{\sigma\vee\tau} \equiv \lambda v.(\mathsf{T}_\sigma\, v \,\|\, \mathsf{T}_\tau\, v) \text{ where } \sigma\vee\tau \neq \omega.
\end{array}$$

Notice the different use of call-by-value variables in the definitions of $\mathsf{T}_{\sigma\to\tau}$ and $\mathsf{T}_{\sigma\vee\tau}$: $\mathsf{T}_{\sigma\to\tau}$ must check that its argument has type $\sigma\to\tau$ which, by $(*)$ above, implies that it has to be convergent. On the other hand, the argument of $\mathsf{T}_{\sigma\vee\tau}$ may reduce to a sum $P+Q$ having type $\sigma\vee\tau$ because P has type σ and Q has type τ but neither σ nor τ can be deduced for $P+Q$. Therefore, it is essential that it is evaluated before the application in parallel of T_σ and T_τ.

The types which can be deduced for R_σ and T_σ are meaningful for their operational behavior. In fact,

- R_σ has exactly the types greater than or equal to σ;
- T_σ has type $\tau\to\rho\to\rho$ only if $\tau\le\sigma$.

This means that R_σ is "the worst" term of type σ and that $\mathsf{T}_\sigma M$ reduces to a value if and only if we can deduce the type σ for M (and this value behaves like the identity combinator).

LEMMA 5.2.

(i) $\vdash_{\mathcal{L}} \mathsf{R}_\sigma : \tau \;\Leftrightarrow\; \sigma\le\tau$;
(ii) $\vdash_{\mathcal{L}} \mathsf{T}_\sigma : \tau\to\gamma\to\delta \;\Leftrightarrow\; \tau\le\sigma \;\&\; \gamma\le\delta$.

Proof. We prove (i) and (ii) simultaneously by induction on σ. We consider only the interesting cases.

$\sigma\equiv\mu\to\nu$:

(i) $\mathsf{R}_{\mu\to\nu} \equiv \lambda x.(\mathsf{T}_\mu x)\,\mathsf{R}_\nu$ and assume $\vdash_{\mathcal{L}} \mathsf{R}_{\mu\to\nu} : \tau$. Then we proceed by a subordinate induction on the structure of τ, the base case being $\tau\equiv\alpha\to\beta$ since $\mathsf{R}_{\mu\to\nu}$ is an abstraction (see Theorem 3.13(ii)). Now, by Theorem 3.13(iii) and (vi), we have $x:\alpha\vdash_{\mathcal{L}} \mathsf{T}_\mu : \gamma\to\delta\to\beta$, $x:\alpha\vdash_{\mathcal{L}} x:\gamma$, and $x:\alpha\vdash_{\mathcal{L}} \mathsf{R}_\nu:\delta$ for some γ and δ. By induction and Theorem 3.13(i) this implies $\gamma\le\mu$, $\delta\le\beta$, $\alpha\le\gamma$, and $\nu\le\delta$; that is, $\alpha\le\mu$ and $\nu\le\beta$. We conclude that $\mu\to\nu\le\alpha\to\beta$.
When $\tau\equiv\alpha\vee\beta$ the thesis follows from the subordinate induction hypothesis. In fact, Theorem 3.15(ii) implies that $\vdash_{\mathcal{L}} \mathsf{R}_{\mu\to\nu}:\alpha$ or $\vdash_{\mathcal{L}} \mathsf{R}_{\mu\to\nu}:\beta$, $\mathsf{R}_{\mu\to\nu}$ being a total value.
The case $\tau\equiv\alpha\wedge\beta$ follows immediately from the subordinate induction hypothesis.

(ii) $\mathsf{T}_{\mu\to\nu}\equiv\lambda v.\mathsf{T}_\nu(v\mathsf{R}_\mu)$ and suppose that $\vdash_{\mathcal{L}} \mathsf{T}_{\mu\to\nu}:\tau\to\gamma\to\delta$. Then we have

$$\begin{array}{ll} & \forall \tau' \in \Theta(\tau),\ v : \tau' \vdash_{\mathcal{L}} \mathsf{T}_\nu(v\mathsf{R}_\mu) : \gamma \to \delta \quad \text{by Theorem 3.13(iv),} \\ \Rightarrow & \forall \tau' \in \Theta(\tau)\ \exists \alpha,\ v : \tau' \vdash_{\mathcal{L}} \mathsf{T}_\nu : \alpha \to \gamma \to \delta\ \&\ v : \tau' \vdash_{\mathcal{L}} v\mathsf{R}_\mu : \alpha \\ & \qquad \text{by Theorem 3.13(vi),} \\ \Rightarrow & \forall \tau' \in \Theta(\tau)\ \exists \alpha,\ \alpha \le \nu\ \&\ \gamma \le \delta\ \&\ \tau' \le \mu \to \alpha \\ & \qquad \text{by induction and Theorem 3.13(i), (vi),} \\ \Rightarrow & \forall \tau' \in \Theta(\tau),\ \tau' \le \mu \to \nu\ \&\ \gamma \le \delta, \\ \Rightarrow & \tau \le \mu \to \nu\ \&\ \gamma \le \delta. \end{array}$$

$\sigma \equiv \mu \vee \nu$:

(ii) $\mathsf{T}_{\mu\vee\nu} \equiv \lambda v.(\mathsf{T}_\mu v \| \mathsf{T}_\nu v)$ and suppose that $\vdash_{\mathcal{L}} \mathsf{T}_{\mu\vee\nu} : \tau \to \gamma \to \delta$. Then we have

$$\begin{array}{ll} & \forall \tau' \in \Theta(\tau),\ v : \tau' \vdash_{\mathcal{L}} (\mathsf{T}_\mu v \| \mathsf{T}_\nu v) : \gamma \to \delta \quad \text{by Theorem 3.13(iv),} \\ \Rightarrow & \forall \tau' \in \Theta(\tau)\ \exists \rho_1, \rho_2,\ v : \tau' \vdash_{\mathcal{L}} \mathsf{T}_\mu v : \rho_1\ \&\ v : \tau' \vdash_{\mathcal{L}} \mathsf{T}_\nu : \rho_2\ \& \\ & \rho_1 \wedge \rho_2 \le \gamma \to \delta \qquad \text{by Theorem 3.13(viii).} \end{array}$$

We assume $\rho_1 \neq \omega$ and $\rho_2 \neq \omega$. $\rho_1 \wedge \rho_2 \le \gamma \to \delta$ implies by Lemma 3.9(i) that there are δ_1, δ_2 such that

$$\delta_1 \wedge \delta_2 = \delta\ \&\ \rho_1 \le \gamma \to \delta_1\ \&\ \rho_2 \le \gamma \to \delta_2.$$

$$\begin{array}{lll} & v : \tau' \vdash_{\mathcal{L}} \mathsf{T}_\mu v : \rho_1, & \\ \Rightarrow & \exists \alpha,\ v : \tau' \vdash_{\mathcal{L}} v : \alpha\ \&\ \vdash_{\mathcal{L}} \mathsf{T}_\mu : \alpha \to \rho_1 & \text{by Theorem 3.13(vi),} \\ \Rightarrow & \vdash_{\mathcal{L}} \mathsf{T}_\mu : \tau' \to \rho_1 & \text{by Theorem 3.13(i)} \\ & & \text{and rule}(\le), \\ \Rightarrow & \vdash_{\mathcal{L}} \mathsf{T}_\mu : \tau' \to \gamma \to \delta_1 & \text{by above and rule } (\le), \\ \Rightarrow & \tau' \le \mu\ \&\ \gamma \le \delta_1 & \text{by induction.} \end{array}$$

Analogously, from $v : \tau' \vdash_{\mathcal{L}} \mathsf{T}_\nu v : \rho_2$ we deduce $\tau' \le \nu$ and $\gamma \le \delta_2$. So we can conclude $\tau \le \mu \vee \nu$ and $\gamma \le \delta_1 \wedge \delta_2$.
The case in which one of ρ_i $(i = 1, 2)$ is equal to ω is similar and simpler. In fact, if $\rho_2 = \omega$ we have $\rho_1 \le \gamma \to \delta$. This allows us to prove $\tau' \le \mu$ and $\gamma \le \delta$ from $v : \tau' \vdash_{\mathcal{L}} \mathsf{T}_\mu v : \rho_1$. So we can conclude once more that $\tau \le \mu \vee \nu$ and $\gamma \le \delta$. □

5.2. Realizability. The aim of this subsection is to prove that

$$\vdash_{\mathcal{L}} M : \omega \to \omega\ \Rightarrow\ M\Downarrow$$

for all closed terms M. As an immediate consequence we have that only types equivalent to ω can be derived for divergent terms.

The proof of this fact requires a double induction, on types and deductions. Following a standard methodology, we split this induction by introducing a "realizability interpretation" of types as sets of closed terms.

DEFINITION 5.3. *We define the mapping* $[\![\cdot]\!] : \mathit{Type} \to \mathcal{P}(\Lambda^0_{+\|})$ *by induction.*

(i) (a) $[\![\omega]\!] = \Lambda^0_{+\|}$;
(b) $[\![\sigma \to \tau]\!] = \{M \in \Lambda^0_{+\|} \mid M\Downarrow\ \&\ \forall N \in [\![\sigma]\!] \Rightarrow MN \in [\![\tau]\!]\}$;
(c) $[\![\sigma \wedge \tau]\!] = \{M \in \Lambda^0_{+\|} \mid M \in [\![\sigma]\!]$ *and* $M \in [\![\tau]\!]\}$;
(d) $[\![\sigma \vee \tau]\!] = \{M \in \Lambda^0_{+\|} \mid M \in [\![\sigma]\!]$ *or* $M \in [\![\tau]\!]$ *or* $\exists N \in [\![\sigma]\!], L \in [\![\tau]\!],\ M \bowtie N + L\}$.

(ii) *If M is open, let $FV(M) = \{x_1, \ldots, x_m, v_1, \ldots, v_n\}$, and $\Gamma(x_i) = \mu_i$ $(1 \leq i \leq m)$, $\Gamma(v_j) = \nu_j$ $(1 \leq j \leq n)$; then*

$$\Gamma \models^r M : \sigma \Leftrightarrow M[N_1/x_1, \ldots, N_m/x_m, L_1/v_1, \ldots, L_n/v_n] \in [\![\sigma]\!]$$

for all $N_i \in [\![\mu_i]\!]$ and $L_j \in [\![\nu_j]\!]$ $(1 \leq i \leq m$ and $1 \leq j \leq n)$.

The correctness of this definition is due to the fact that all types are inhabited by some closed term; in fact, Theorem 5.5 will imply that $\models^r \mathsf{R}_\sigma : \sigma$ for all types σ.

The following lemma states some key properties of our realizability interpretation.

LEMMA 5.4. *Let $M, N, W \in \Lambda^0_{+\|}$. Then*

(i) $M \in [\![\sigma]\!]$ & $M \bowtie N \Rightarrow N \in [\![\sigma]\!]$;
(ii) $M \in [\![\sigma]\!] \Rightarrow M\|N \in [\![\sigma]\!]$;
(iii) $M \in [\![\sigma]\!]$ & $N \in [\![\sigma]\!] \Leftrightarrow M + N \in [\![\sigma]\!]$;
(iv) $W \in [\![\sigma \vee \tau]\!]$ & $W \in \mathsf{TVal} \Rightarrow W \in [\![\sigma]\!]$ *or* $W \in [\![\tau]\!]$;
(v) $M \in [\![\sigma]\!]$ & $\sigma \leq \tau \Rightarrow M \in [\![\tau]\!]$;
(vi) $M \in [\![\sigma]\!]$ & $\sigma \neq \omega \Rightarrow M{\Downarrow}$.

Proof. We prove points (i)–(iii) of this lemma by induction on σ. The case $\sigma \equiv \omega$ is always trivial. The cases $\sigma \equiv \tau \wedge \rho$ and $\sigma \equiv \tau \vee \rho$ with $M \in [\![\tau]\!] \cup [\![\rho]\!]$ $(N \in [\![\tau]\!] \cup [\![\rho]\!])$ immediately follow from the induction hypothesis. Therefore, the proofs of these cases are omitted.

(i) Case $\sigma \equiv \tau \to \rho$. $M{\Downarrow}$ implies $N{\Downarrow}$ by Theorem 2.12(ii); moreover,

$$\begin{array}{lll} & M \in [\![\sigma]\!] & \\ \Rightarrow & \forall L \in [\![\tau]\!],\ ML \in [\![\rho]\!] & \text{by definition,} \\ \Rightarrow & \forall L \in [\![\tau]\!],\ NL \in [\![\rho]\!] & \text{by induction since } ML \bowtie NL \text{ by Lemma 2.8(i),} \\ \Rightarrow & N \in [\![\sigma]\!] & \text{by definition.} \end{array}$$

Case $\sigma \equiv \tau \vee \rho$ and $M \bowtie P + Q$ for some $P \in [\![\tau]\!]$ and $Q \in [\![\rho]\!]$. Therefore, $N \bowtie P + Q$ and we are done.

(ii) Case $\sigma \equiv \tau \to \rho$. $M{\Downarrow}$ implies $M\|N{\Downarrow}$; moreover,

$$\begin{array}{lll} & M \in [\![\sigma]\!] & \\ \Rightarrow & \forall L \in [\![\tau]\!],\ ML \in [\![\rho]\!] & \text{by definition,} \\ \Rightarrow & \forall L \in [\![\tau]\!], N \in \Lambda^0_{+\|},\ ML\|NL \in [\![\rho]\!] & \text{by induction,} \\ \Rightarrow & \forall L \in [\![\tau]\!], N \in \Lambda^0_{+\|},\ (M\|N)L \in [\![\rho]\!] & \text{by (i) since } (M\|N)L \rhd ML\|NL \\ & & \text{by rule } (\|_{app})', \\ \Rightarrow & \forall N \in \Lambda^0_{+\|}, M\|N \in [\![\sigma]\!] & \text{by definition.} \end{array}$$

Case $\sigma \equiv \tau \vee \rho$ and $M \bowtie P + Q$ for some $P \in [\![\tau]\!]$ and $Q \in [\![\rho]\!]$. $M \bowtie P + Q$ implies $M\|N \bowtie (P+Q)\|N$ by Lemma 2.8(i) and $(P+Q)\|N \rhd P\|N + Q\|N$ by rule $(+\|)'$ for all $N \in \Lambda^0_{+\|}$. By induction, $P\|N \in [\![\tau]\!]$ and $Q\|N \in [\![\rho]\!]$, so we conclude that $M\|N \in [\![\sigma]\!]$ for all $N \in \Lambda^0_{+\|}$.

(iii) $(\Rightarrow)$.
Case $\sigma \equiv \tau \to \rho$.

$$\begin{array}{lll} & M, N \in [\![\sigma]\!] & \\ \Rightarrow & \forall L \in [\![\tau]\!],\ ML, NL \in [\![\rho]\!] & \text{by definition,} \\ \Rightarrow & \forall L \in [\![\tau]\!],\ ML + NL \in [\![\rho]\!] & \text{by induction,} \\ \Rightarrow & \forall L \in [\![\tau]\!],\ (M+N)L \in [\![\rho]\!] & \text{by (i) since } (M+N)L \rhd ML + NL \\ & & \text{by rule } (+_{app})', \\ \Rightarrow & M + N \in [\![\sigma]\!] & \text{by definition.} \end{array}$$

Case $\sigma \equiv \tau \vee \rho$ and $M \bowtie M_0 + M_1$, $N \bowtie N_0 + N_1$ for some $M_0, N_0 \in [\![\tau]\!]$ and $M_1, N_1 \in [\![\rho]\!]$. From the induction hypothesis $M_0 + N_0 \in [\![\tau]\!]$ and $M_1 + N_1 \in [\![\rho]\!]$; therefore,

$$M + N \bowtie (M_0 + N_0) + (M_1 + N_1) \in [\![\tau \vee \rho]\!].$$

($\Leftarrow$) Case $\sigma \equiv \tau \to \rho$.

$$\begin{array}{lll} & M + N \in [\![\sigma]\!] & \\ \Rightarrow & \forall L \in [\![\tau]\!],\ (M+N)L \in [\![\rho]\!] & \text{by definition,} \\ \Rightarrow & \forall L \in [\![\tau]\!],\ ML + NL \in [\![\rho]\!] & \text{by (i) since } (M+N)L \rhd ML + NL \\ & & \text{by rule } (+_{app})', \\ \Rightarrow & \forall L \in [\![\tau]\!],\ ML, NL \in [\![\rho]\!] & \text{by induction} \\ \Rightarrow & M, N \in [\![\sigma]\!] & \text{by definition.} \end{array}$$

Case $\sigma \equiv \tau \vee \rho$ and $M + N \bowtie P + Q$ for some $P \in [\![\tau]\!]$ and $Q \in [\![\rho]\!]$. If $M \bowtie P$ and $N \bowtie Q$ (or vice versa) the thesis follows from (i). Otherwise, by Lemma 2.8(ii) there exist M_0, M_1, N_0, N_1 such that $M \bowtie M_0 + M_1$, $N \bowtie N_0 + N_1$ and $P \bowtie M_0 + N_0$, $Q \bowtie M_1 + N_1$. By induction, $M_0, N_0 \in [\![\tau]\!]$ and $M_1, N_1 \in [\![\rho]\!]$. So we conclude that $M \in [\![\tau \vee \rho]\!]$ and $N \in [\![\tau \vee \rho]\!]$ by definition.

(iv) It suffices to observe that $W \in \mathsf{TVal}$ implies that $W \bowtie M + N$ is impossible for any M, N.

(v) This proof is by induction on the definition of $\leq$. In the case $\sigma \vee \sigma \leq \sigma$, use (iii)($\Rightarrow$).

We consider the case $(\sigma \vee \tau) \wedge \rho \leq (\sigma \wedge \rho) \vee (\tau \wedge \rho)$ when $M \bowtie P + Q$, $P \in [\![\sigma]\!]$, $Q \in [\![\tau]\!]$, and $M \in [\![\rho]\!]$. $M \in [\![\rho]\!]$ implies $P \in [\![\rho]\!]$ and $Q \in [\![\rho]\!]$ by (i) and (iii)($\Leftarrow$). Therefore, we have $P \in [\![\sigma \wedge \rho]\!]$ and $Q \in [\![\tau \wedge \rho]\!]$, so we can conclude $M \in [\![(\sigma \wedge \rho) \vee (\tau \wedge \rho)]\!]$.

(vi) This proof is by induction on σ, taking into account that $\sigma \equiv \tau \vee \rho$ and $\sigma \neq \omega$ imply $\tau \neq \omega$ and $\rho \neq \omega$. □

As expected, realizability coincides with derivability in $\vdash_{\mathcal{L}}$, and this implies in turn that we can assure convergence for all closed terms typable by $\omega \to \omega$.

THEOREM 5.5 (realizability theorem).

$$\forall \Gamma, \sigma, M,\ \Gamma \models^r M : \sigma \ \Leftrightarrow\ \Gamma \vdash_{\mathcal{L}} M : \sigma.$$

Proof. Let $FV(M) = \{x_1, \ldots, x_m, v_1, \ldots, v_n\}$ and $\Gamma = \{x_1{:}\mu_1, \ldots,\ x_m{:}\mu_m,\ v_1{:}\nu_1, \ldots, v_n{:}\nu_n\}$. Let us choose $N_1, \ldots, N_m$ and $L_1, \ldots, L_n$ as in Definition 5.3(ii) and put

$$U^* \equiv U[N_1/x_1, \ldots, N_m/x_m, L_1/v_1, \ldots, L_n/v_n].$$

It suffices to show that $M^* \in [\![\sigma]\!]$ if and only if $\Gamma \vdash_{\mathcal{L}} M : \sigma$.

($\Leftarrow$) The proof is by induction on deductions. We consider only the interesting cases.

Case ($\to \mathrm{I}_v$). Then $M \equiv \lambda v.P$ for some P, $\sigma \equiv \tau \to \rho$ and $\Gamma, v : \tau' \vdash P : \rho$ has been derived for all $\tau' \in \Theta(\tau)$. Let $L \in [\![\tau']\!]$ for some $\tau' \in \Theta(\tau)$. $L \in [\![\tau']\!]$ implies $L \Downarrow$ by Lemma 5.4(vi) since $\tau' \neq \omega$. Therefore, by Corollary 2.10, there are $V_1, \ldots, V_k \in \mathsf{Val}$ such that $L \rhd \sum_{i=1}^{k} V_i$ and $(\lambda v.P)L \rhd \sum_{i=1}^{k} (\lambda v.P)V_i$. Notice that by Lemma 5.4(i) and (iii), $\forall i \leq k,\ V_i \in [\![\tau']\!]$.

From the induction hypothesis for all $i \leq n$, $P[V_i/v]^* \in [\![\rho]\!]$; there is no loss of generality in supposing that v does not occur in $N_1, \ldots, N_m, L_1, \ldots, L_n$ so that $(P[V_i/v])^* \equiv P^*[V_i/v]$. We have $(\lambda v.P)^* V_i \equiv (\lambda v.P^*)V_i$. Now $(\lambda v.P^*)V_i \rhd P^*[V_i/v]$ if $V_i \in \mathsf{TVal}$ and $(\lambda v.P^*)V_i \rhd \sum_{j=1}^{h}(P^*[V_i/v]\|(\lambda v.P^*)V'_j)$, where $\mathcal{R}(V_i, 1) = \{V'_1, \ldots, V'_h\}$ otherwise. In both cases it follows that $(\lambda v.P)^* V_i \in [\![\rho]\!]$ by Lemma 5.4(i), (ii), (iii).

$$\begin{array}{lcll} \forall i \leq k,\ M^*V_i \in [\![\rho]\!] & \Rightarrow & \sum_{i=1}^{k} M^*V_i \in [\![\rho]\!] & \text{by Lemma 5.4(iii),} \\ & \Rightarrow & M^*L \in [\![\rho]\!] & \text{by Lemma 5.4(i)} \\ & & & \text{since } M^*L \rhd \sum_{i=1}^{k} M^*V_i \\ & & & \text{by construction.} \end{array}$$

So we conclude $M^* \in [\![\sigma]\!]$ by the arbitrariness of the computable term L.

Case ($\|$ I). Then $M \equiv P\|Q$ for some P, Q, $\sigma \equiv \tau \wedge \rho$ and, say, $\Gamma \vdash P : \tau$ and $\Gamma \vdash Q : \rho$ have been derived. From the induction hypothesis, $P^* \in [\![\tau]\!]$ and $Q^* \in [\![\rho]\!]$, so that by Lemma 5.4(ii), $P^*\|Q^* \in [\![\tau]\!]$ and $P^*\|Q^* \in [\![\rho]\!]$, which imply by definition $(P\|Q)^* \in [\![\sigma]\!]$.

($\Rightarrow$) This proof is by induction on σ. The only interesting case is when $\sigma \equiv \tau \to \rho$.

$$\begin{array}{lcll} M^* \in [\![\sigma]\!] & \Rightarrow & \forall N \in [\![\tau]\!],\ M^*N \in [\![\rho]\!] & \text{by definition,} \\ & \Rightarrow & M^*\mathsf{R}_\tau \equiv (M\mathsf{R}_\tau)^* \in [\![\rho]\!] & \text{since } \mathsf{R}_\tau \in [\![\tau]\!] \text{ by } (\Leftarrow) \\ & \Rightarrow & \Gamma \vdash_{\mathcal{L}} M\mathsf{R}_\tau : \rho & \text{by induction} \\ & \Rightarrow & \Gamma \vdash_{\mathcal{L}} M : \tau \to \rho & \text{by Theorem 3.13(vi) and} \\ & & & \text{Lemma 5.2(i).} \quad \square \end{array}$$

The main result of this subsection is the characterization of convergent terms by the type $\omega \to \omega$ (see also Theorem 3.23).

COROLLARY 5.6.

(i) $\forall M \in \Lambda^0_{+\|},\ \vdash_{\mathcal{L}} M : \omega \to \omega \Rightarrow M{\Downarrow}$.

(ii) $\forall M \in \Lambda^0_{+\|},\ M{\Uparrow}\ \&\ \vdash_{\mathcal{L}} M : \sigma \Rightarrow \sigma = \omega$.

As already stated, the characterization of types which can be deduced for divergent terms given in Corollary 5.6(ii) allows us to prove that some inclusions in the model are proper.

COROLLARY 5.7.

(i) $\exists M \in \Lambda_{+\|}, V \in \mathsf{Val}$, *$\sigma$ such that* $[\forall \sigma' \in \Theta(\sigma),\ v : \sigma' \vdash_{\mathcal{L}} M : \tau]\ \&\ \vdash_{\mathcal{L}} V : \sigma$ *but* $\not\vdash_{\mathcal{L}} M[V/v] : \tau$;

(ii) $\exists M \in \Lambda_{+\|}, V \in \mathsf{Val}$, *$\tau$ such that* $\vdash_{\mathcal{L}} (\lambda v.M)V : \tau$ *but* $\not\vdash_{\mathcal{L}} M[V/v] : \tau$;

(iii) $\exists L, M, N \in \Lambda_{+\|}$ *such that* $L(M + N) \sqsubset^{\mathcal{L}} LM + LN$;

(iv) $\exists L, M, N \in \Lambda_{+\|}$ *such that* $LM\|LN \sqsubset^{\mathcal{L}} L(M\|N)$.

Proof.

(i) An example is $M \equiv v\mathbf{I}\Delta\Delta\|v\Delta\mathbf{I}\Delta$, $V \equiv \Delta\|(\mathbf{K} + \mathbf{O})$, and $\sigma = \sigma_1 \vee \sigma_2$, where $\sigma_1 \equiv \rho \to \omega \to \rho$, $\sigma_2 \equiv \omega \to \rho \to \rho$, $\rho \equiv \tau \to \tau$, and $\tau \equiv (\omega^2 \to \omega) \to \omega \to \omega$. We can easily check that (1) $\vdash_{\mathcal{L}} \mathbf{I} : \rho$, (2) $\vdash_{\mathcal{L}} \Delta : \tau$, (3) $\vdash_{\mathcal{L}} \mathbf{K} : \sigma_1$, and (4) $\vdash_{\mathcal{L}} \mathbf{O} : \sigma_2$. From (1) and (2) we obtain (5) $v : \sigma_1 \vdash_{\mathcal{L}} v\mathbf{I}\Delta\Delta : \tau$ and (6) $v : \sigma_2 \vdash_{\mathcal{L}} v\Delta\mathbf{I}\Delta : \tau$, which imply, respectively, $v : \sigma_1 \vdash_{\mathcal{L}} M : \tau$ and $v : \sigma_2 \vdash_{\mathcal{L}} M : \tau$. Using (3) and (4) we derive $\vdash_{\mathcal{L}} \mathbf{K} + \mathbf{O} : \sigma_1 \vee \sigma_2$, which implies $\vdash_{\mathcal{L}} V : \sigma_1 \vee \sigma_2$. But type τ cannot be deduced for $M[V/v]$. In fact, it is easy to verify that $M[V/v]$ diverges, since it reduces to $\Omega\|\Omega\|\Omega\mathbf{I}\Delta\|\Omega$. Therefore, by Corollary 5.6(ii) $M[V/v]$ has only types equivalent to ω.

(ii) This proof is immediate from (i).

(iii) An example is $L \equiv \lambda x.(x\mathbf{I}\Delta\Delta \| x\Delta\mathbf{I}\Delta)$, $M \equiv \mathbf{K}$, and $N \equiv \mathbf{O}$. Analogously to (5) and (6) (in the proof of (i)) we have (7) $x : \sigma_1 \vdash_{\mathcal{L}} x\mathbf{I}\Delta\Delta : \tau$ and (8) $x : \sigma_2 \vdash_{\mathcal{L}} x\Delta\mathbf{I}\Delta : \tau$, which imply (9) $\vdash_{\mathcal{L}} L : \sigma_1 \to \tau$ and (10) $\vdash_{\mathcal{L}} L : \sigma_2 \to \tau$. Then from (3), (4), (9), and (10) we conclude $\vdash_{\mathcal{L}} LM + LN : \tau$, but this type cannot be deduced for $L(M + N)$. Really, $L(\mathbf{K} + \mathbf{O})$ reduces to $\Omega\|\Omega$, which being divergent has only types equivalent to ω.

(iv) We choose $L \equiv \lambda x.(x\mathbf{I}\Delta\Delta + x\Delta\mathbf{I}\Delta)$, $M \equiv \mathbf{K}$, and $N \equiv \mathbf{O}$. Using (7) and (8) (from the proof of (iii)) we have $\vdash_{\mathcal{L}} L : \sigma_1 \wedge \sigma_2 \to \tau$. (3) and (4) (from the proof of (i)) imply $\vdash_{\mathcal{L}} \mathbf{K}\|\mathbf{O} : \sigma_1 \wedge \sigma_2$. Then $\vdash_{\mathcal{L}} L(M\|N) : \tau$, but this type cannot be deduced for $LM\|LN$. The reason is again that this term diverges, since it reduces to $\Omega\|\Omega$. □

Notice that Corollary 5.7(i) says that we cannot relax the condition on the premises of rule ($\vee$ E) (discussed on p. 1398), allowing W to be a partial value. This does not mean that our calculus distinguishes internally between partial and total values. In fact, we do not have a type (and therefore a test term by the following full abstraction theorem) that characterizes all total values, as type $\omega \to \omega$ characterizes all convergent terms. The best we can do is the statement of Theorem 3.15(ii).

As suggested by one of the referees, some further remarks are in order about the theory of the model. In fact, equivalence classes of terms with respect to $\simeq^{\mathcal{L}}$ build a distributive lattice in which $+$ is the meet and $\|$ is the join. Moreover, Ω and all divergent terms are the bottom, since only types equivalent to ω can be deduced for them. Finally, the lattice has a top, namely, the equivalence class of the term $(\lambda xy.xx)(\lambda xy.xx)$ (usually called the "ogre"). We recall that $(\lambda xy.xx)(\lambda xy.xx)$ is convertible to the fixed-point of $\mathbf{K}$, i.e., to $\mathbf{YK}$, using the standard β-rule. Moreover, it is unsolvable of infinite order, since it reduces to $\lambda z_1 \dots z_n.(\lambda xy.xx)(\lambda xy.xx)$ for any $n \geq 0$ using the standard β-rule. To prove that *Type* is the interpretation of the "ogre," in the remaining part of this section we will show that $\vdash_{\mathcal{L}} (\lambda xy.xx)(\lambda xy.xx) : \sigma$ for all types σ.

Define, for every $n \geq 1$, the following types;

$$\begin{array}{ll} \sigma_1 \equiv \omega, & \\ \sigma_{n+1} \equiv \tau_n \wedge \sigma_n, & \qquad \tau_n \equiv \sigma_n \to \omega^n \to \omega. \end{array}$$

LEMMA 5.8.

(i) *For every* $n \geq 1$, $\vdash_{\mathcal{L}} \lambda xy.xx : \tau_n$.

(ii) *For every* $n \geq 1$, $\vdash_{\mathcal{L}} \lambda xy.xx : \sigma_n$.

Proof.

(i) We distinguish two cases.

Case $n = 1$.

$$\dfrac{\dfrac{\vdash xx : \omega}{\vdash \lambda y.xx : \omega \to \omega}\ (\to \mathrm{I}_n)}{\vdash \lambda xy.xx : \omega \to \omega \to \omega}\ (\to \mathrm{I}_n).$$

Case $n > 1$.

$$\dfrac{\dfrac{\dfrac{\dfrac{x{:}\sigma_n \vdash x : \tau_{n-1} \wedge \sigma_{n-1}}{x{:}\sigma_n \vdash x : \tau_{n-1}}\ (\leq) \quad \dfrac{x{:}\sigma_n \vdash x : \tau_{n-1} \wedge \sigma_{n-1}}{x{:}\sigma_n \vdash x : \sigma_{n-1}}\ (\leq)}{x{:}\sigma_n \vdash xx : \omega^{n-1} \to \omega}\ (\to \mathrm{E})}{x{:}\sigma_n \vdash \lambda y.xx : \omega^n \to \omega}\ (\to \mathrm{I}_n)}{\vdash \lambda xy.xx : \sigma_n \to \omega^n \to \omega}\ (\to \mathrm{I}_n).$$

(ii) The case $n = 1$ is trivial. If $n > 1$, then $\sigma_n = \bigwedge_{i=1}^{n-1} \tau_i$, so the thesis follows immediately from (i). □

THEOREM 5.9.

(i) *For every* $n \geq 1$, $\vdash_{\mathcal{L}} (\lambda xy.xx)(\lambda xy.xx) : \omega^n \to \omega$.

(ii) *For all types* σ, $\vdash_{\mathcal{L}} (\lambda xy.xx)(\lambda xy.xx) : \sigma$.

Proof.

(i) Just consider the derivations, for $n \geq 1$,

$$\frac{\dfrac{\text{(by Lemma 5.8(i))}}{\vdash \lambda xy.xx : \tau_n} \qquad \dfrac{\text{(by Lemma 5.8(ii))}}{\vdash \lambda xy.xx : \sigma_n}}{\vdash (\lambda xy.xx)(\lambda xy.xx) : \omega^n \to \omega} \ (\to \mathrm{E}),$$

since $\tau_n \equiv \sigma_n \to \omega^n \to \omega$.

(ii) This proof is immediate from (i) and 3.3. □

5.3. Full abstraction. To establish the main result of this paper we use the discriminability power of test terms.

THEOREM 5.10. *Let* M *be a closed term. Then* $\mathsf{T}_\sigma M\Downarrow \ \Leftrightarrow \ \vdash_{\mathcal{L}} M : \sigma$.

Proof. If $\mathsf{T}_\sigma M\Downarrow$ then, by Theorem 3.23, $\vdash_{\mathcal{L}} \mathsf{T}_\sigma M : \omega \to \omega$. By Theorem 3.13(vi) it follows that $\mathsf{T}_\sigma : \rho \to \omega \to \omega$ for some ρ such that $\vdash_{\mathcal{L}} M : \rho$. By Lemma 5.2(ii) it has to be true that $\rho \leq \sigma$, so that $\vdash_{\mathcal{L}} M : \sigma$ using rule $(\leq)$. Vice versa, $\vdash_{\mathcal{L}} M : \sigma$ implies $\vdash_{\mathcal{L}} \mathsf{T}_\sigma M : \omega \to \omega$ by Lemma 5.2(ii), so we conclude $\mathsf{T}_\sigma M\Downarrow$ by Corollary 5.6(i). □

THEOREM 5.11 (full abstraction).

$$M \sqsubseteq^{\mathcal{L}} N \Leftrightarrow M \sqsubseteq^{\mathcal{O}} N.$$

Proof. $(\Rightarrow)$ (adequacy) Since $\sqsubseteq^{\mathcal{L}}$ is a precongruence, $M \sqsubseteq^{\mathcal{L}} N$ implies that, for any context $C[\,]$ closing both M and N, if $\vdash_{\mathcal{L}} C[M] : \omega \to \omega$ then $\vdash_{\mathcal{L}} C[N] : \omega \to \omega$. It follows that

$$\begin{aligned} C[M]\Downarrow &\Rightarrow \vdash_{\mathcal{L}} C[M] : \omega \to \omega \text{ by Theorem 3.23} \\ &\Rightarrow \vdash_{\mathcal{L}} C[N] : \omega \to \omega \Rightarrow C[N]\Downarrow \text{ by Corollary 5.6(i).} \end{aligned}$$

$(\Leftarrow)$ (completeness) $M \not\sqsubseteq^{\mathcal{L}} N \Rightarrow \exists \Gamma, \sigma,\ \Gamma \vdash_{\mathcal{L}} M : \sigma \ \&\ \Gamma \not\vdash_{\mathcal{L}} N : \sigma$. Let $FV(MN) = \{\chi_i \mid 1 \leq i \leq n\}$, $\Gamma = \{\chi_i : \tau_i \mid 1 \leq i \leq n\}$, and $\tau = \tau_1 \to \cdots \to \tau_n \to \sigma$; then $\vdash_{\mathcal{L}} \lambda\chi_1 \ldots \chi_n.M : \tau$ and $\not\vdash_{\mathcal{L}} \lambda\chi_1 \ldots \chi_n.N : \tau$ by Theorem 3.13(iii), (iv). Therefore, choosing $C[\,] = \mathsf{T}_\tau(\lambda\chi_1 \ldots \chi_n.[\,])$, we have by Theorem 5.10 that $C[M]\Downarrow$ and $C[N]\Uparrow$, which imply $M \not\sqsubseteq^{\mathcal{O}} N$. □

A natural question, posed by one of the referees, is whether $\simeq^{\mathcal{L}}$ on the sub-calculus without + coincides with Boudol's equivalence [19]. The answer is negative, since our calculus is more discriminating also for pure λ-terms. For example, Boudol equates $\lambda x.xx$ and its η-expansion $\lambda x.x(\lambda y.xy)$; this is proved in [61]. Instead, in our calculus with +, $\lambda x.xx$ is strictly better than $\lambda x.x(\lambda y.xy)$. This is shown by the fact that $\vdash_{\mathcal{L}} \lambda x.xx : \sigma \wedge (\sigma \to \tau) \to \tau$, while $\lambda x.x(\lambda y.xy)$ does not have this type, where $\sigma \equiv (\mu \to \omega \to \mu) \vee (\omega \to \mu \to \mu)$, $\tau \equiv \omega \to \mu$, and $\mu \equiv \omega \to \omega$. This can be checked by verifying that the test term corresponding to $\sigma \wedge (\sigma \to \tau) \to \tau$ converges when applied to $\lambda x.xx$ and diverges when applied to $\lambda x.x(\lambda y.xy)$. Another way of showing this is to consider the application of these terms to $P + Q$, where $P \equiv \lambda v.v(v\mathbf{O})$ and $Q \equiv \lambda yx.x\Omega$. In fact, we have that $(\lambda x.xx)(P + Q)$ converges,

while $(\lambda x.x(\lambda y.xy))(P+Q)$ diverges. In Figure 3 we show the whole reduction tree of $(\lambda x.xx)(P+Q)$ and an infinite reduction path out of $(\lambda x.x(\lambda y.xy))(P+Q)$.

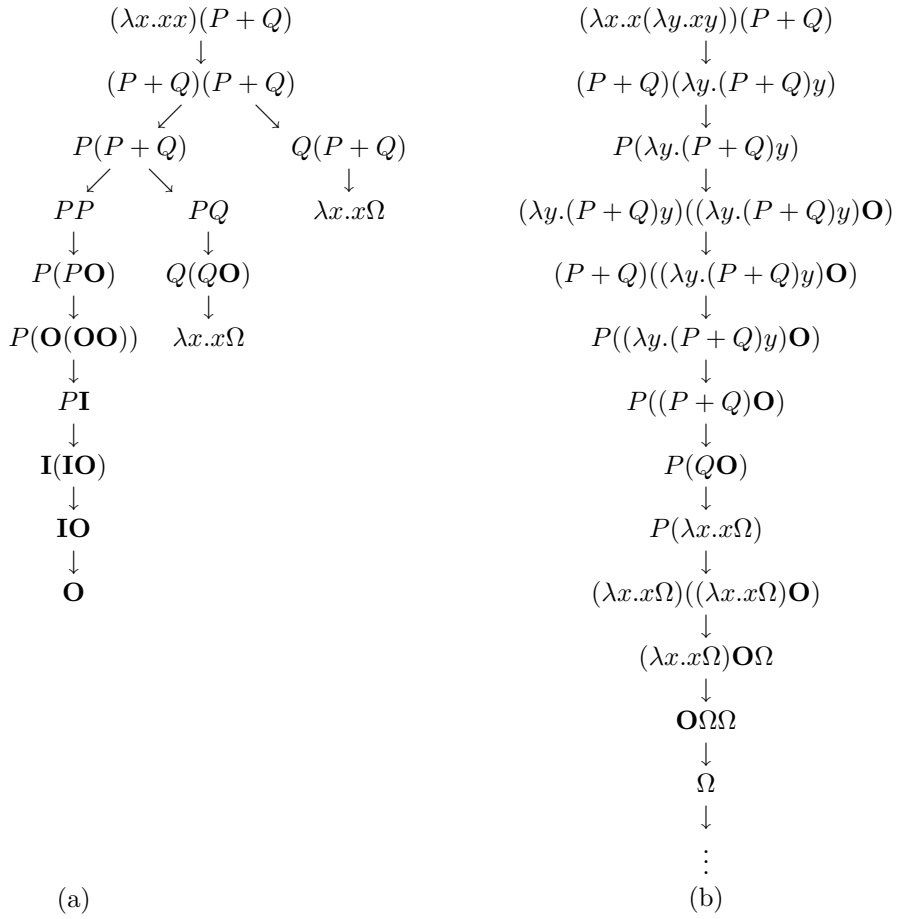


FIG. 3. (a) *The reduction tree of* $(\lambda x.xx)(P+Q)$. (b) *An infinite reduction path out of* $(\lambda x.x(\lambda y.xy))(P+Q)$.

Really, we strongly conjecture that the restriction of $\simeq^{\mathcal{L}}$ to the pure λ-calculus coincides with the equality of Lévy–Longo trees associated with λ-terms. This equivalence relation between λ-terms, which we denote by $\simeq^{\mathcal{T}}$, was defined in [44].

From one side, Sangiorgi in [61] proves that using a set of well-formed operators and comparing terms through bisimulation, one always obtains an equivalence relation (we call $\simeq^{\mathcal{OP}}$) which includes $\simeq^{\mathcal{T}}$. Indeed, we have that $M \simeq^{\mathcal{T}} N$ implies $M \simeq^{\mathcal{OP}} N$ for all λ-terms M, N. Now our concurrent λ-calculus respects the conditions on well-formed operators of [61]. Moreover, we compare terms through contexts, which equate in our case more than bisimulation. For example, we equate $\mathbf{I}+\Omega$ and Ω, which are not bisimilar. Then we conclude that $M \simeq^{\mathcal{T}} N$ implies $M \simeq^{\mathcal{L}} N$ for all λ-terms M, N.

From the other side, given two different Lévy–Longo trees, we believe that there is always a type which can be deduced only for one of the corresponding terms. Note

that we cannot use Sangiorgi's result that the nondeterministic choice is sufficient to obtain the discriminating power of Lévy–Longo trees, since we compare terms by means of contexts instead of using bisimulation.

The coincidence of $\mathcal{L}$ and $\mathcal{T}$ would be another argument showing the robustness of the theory induced by $\simeq^{\mathcal{T}}$. In fact, the same theory is also induced by:

- the encoding of λ-calculus into π-calculus [61];
- Plotkin–Scott–Engeler models [54];
- contexts with multiplicities [20].

Lastly, we want to discuss the negative results of [64]. In that paper, Sieber considers a call-by-value version of PCF enriched with a nondeterministic choice operator (he calls this language PCF_{nv}). He proves that for this language the full abstraction of the Smyth powerdomain semantics fails in an irreparable way. Indeed, there is no extension of PCF_{nv} by computable operators, for which this semantics is fully abstract.

Notice that PCF_{nv} is a typed language, while our concurrent λ-calculus is type-free and we use types only to describe its semantics. Indeed, it is not surprising that an untyped calculus has a stronger power in discriminating internally between terms than a typed one. In particular, Böhm's theorem [15] (10.4) does not hold for the simply typed λ-calculus.

The same phenomenon happens here, since one can easily discriminate the terms which are at the basis of Sieber's proof. Omitting types, these terms are

$$\begin{aligned} M_1 &= \lambda f.\ \mathbf{if}\ f(\lambda x.0)\ \mathbf{then}\ f(\lambda x.\Omega)\ \mathbf{else}\ \Omega\ \mathbf{fi}, \\ M_2 &= \lambda f.\ \mathbf{if}\ f(\lambda x.0)\ \mathbf{and}\ f(\lambda x.\Omega)\ \mathbf{then}\ 0\ \mathbf{else}\ \Omega\ \mathbf{fi}, \end{aligned}$$

where 0 is interpreted as "true" and all other integers as "false." We can encode this example in the pure λ-calculus with the η-reduction rule using Church's numerals. Recall that $\mathbf{O}$ represents Church's zero and that $\mathbf{I}$ is η-convertible to Church's one. If we denote this translation by $(\ )^\dagger$, it is easy to check that a correct choice is the following:

$$(\ \mathbf{if}\ A\ \mathbf{then}\ B\ \mathbf{else}\ C\ \mathbf{fi})^\dagger = A^\dagger\mathbf{O}(\mathbf{K}B^\dagger)C^\dagger, \qquad (A\ \mathbf{and} B)^\dagger = A^\dagger\mathbf{O}B^\dagger,$$

since for Church's numerals it holds that $(\mathbf{n}+\mathbf{1})\mathbf{O} \xrightarrow{*} \mathbf{O}$. Therefore we have

$$\begin{aligned} M_1^\dagger &= \lambda f.f(\lambda x.\mathbf{O})\mathbf{O}(\mathbf{K}(f(\lambda x.\Omega)))\Omega, \\ M_2^\dagger &= \lambda f.f(\lambda x.\mathbf{O})\mathbf{O}(f(\lambda x.\Omega))\mathbf{O}(\mathbf{KO})\Omega. \end{aligned}$$

Now the pure λ-calculus easily semi-separates these terms. It suffices to choose $F \equiv \lambda x_1 \dots x_6.x_6$, since $M_1^\dagger F \xrightarrow{*} \mathbf{O}$ and $M_2^\dagger F \xrightarrow{*} \Omega$. It is clear that PCF_{nv} does not allow us to define a well-typed operator which behaves like F. Really, F applied to any term returns $\lambda x_1 \dots x_5.x_5$, which does not have the type of integers.

6. Related work and conclusions. The literature related to the present work has mostly been quoted in the introduction. A research direction, which has not yet been mentioned, is that initiated by Milner in [50], and developed by Sangiorgi in [61]. The idea is to consider the π-calculus as the basic model, and to study lazy λ-calculus via encodings into the π-calculus itself. A comparative study and a survey of these investigations is [43]. Although related, this approach seems orthogonal to that which is developed in the present paper, where the nondeterministic lazy λ-calculus is studied with respect to a denotational model via a type assignment system. In spite

of this, as remarked on p. 1415, we conjecture that the theory of this encoding and that of our model coincide.

It is interesting to compare our full abstraction result with the negative results of [39] and [64]. Both papers deal with typed λ-calculus (actually PCF).

Jim and Meyer show in [39] that any denotational semantics which is adequate for PCF, and in which a certain simple boolean functional exists, cannot be fully abstract for extensions of PCF satisfying the context lemma. This boolean functional is not Scott's continuous, but it is stably continuous. So it follows that there is no extension of PCF satisfying the context lemma for which the stable domains are fully abstract. Actually, we consider Scott's continuous functions and moreover our calculus does not satisfy the context lemma. For example, we distinguish between $\mathbf{I}+\Delta$ and $\lambda x.(x+xx)$, which clearly have the same applicative behavior.

Sieber [64] adds call-by-value and nondeterminism to PCF. He explains why the nondeterministic extensions of call-by-name λ-calculus studied in [9, 10] need the powerdomain functors only at ground types. Instead, call-by-value and nondeterminism also require powerdomains for function types. Moreover, Sieber shows that no fully abstract model of demonic nondeterminism and call-by-value can be given in the typed case. The counterexample he considers, however, has no counterpart in the type-free setting (see the discussion at the end of section 5).

A further point concerns previous work by the present authors. In [45] the operational and denotational semantics of a call-by-name classical λ-calculus enriched with erratic choice is studied introducing an extension of the notion of Böhm tree.

In [25] classical λ-calculus is extended with both a nondeterministic choice operator and a parallel operator. A type assignment system using intersection and union types and denotational models of this calculus are investigated. In particular, the local structure of the resulting filter model is studied against a Morris-style (see [52]) operational semantics. The main differences with the present work are the classical operational semantics (that is, evaluation under abstraction is allowed) and the absence of call-by-value abstraction. Notably, full abstraction is a distinctive feature of the present calculus with respect to that studied in [25].

Pursuing further the study of concurrent λ-calculus, in [8] we present an operational semantics in which the parallel operator still retains an angelic behavior, while the nondeterministic operator has a finer semantics than the demonic one. The system admits a model which involves a new powerdomain operator close to the convex powerdomain functor, but staying inside the category of distributive prime algebraic lattices.

Finally, in [27], ideas coming from the present paper are used to define a calculus of higher-order processes including communication primitives. For this calculus a program logic in the form of a type assignment system is introduced. The induced filter model turns out to be fully abstract with respect to the operational preorder. This is closely related, and should be compared, to [35].

Summarizing, the main achievement of the present study is the definition of an operational semantics which assesses in a correct and effective way features like parallelism, nondeterminism, and call-by-value which have not received, to the knowledge of the authors, a convincing treatment within a unique comprehensive system.

Moreover, the abstract description of operational semantics by means of the logic equivalence, which is guaranteed by the full abstraction theorem, shows that an elegant correspondence between the operators in the term syntax and the logical connectives is at the basis of the whole construction.

The logical model is a simple distributive lattice enriched with an operation which interprets functional application. The simplicity of this structure and the correspondence between (total) values and prime elements of the lattice are also remarkable.

Acknowledgments. The present version of this paper has been deeply influenced by comments and remarks of two anonymous referees. In particular, the addition of discussions about motivations, comparisons with the existing literature, justifications of the choices made, and the addition of meaningful examples improved the presentation of the paper. Therefore, the authors feel strongly indebted to the referees.

REFERENCES

[1] M. Abadi, *A semantics for static type inference in a nondeterministic language*, Inform. and Comput., 109 (1994), pp. 300–306.

[2] S. Abramsky, *On semantic foundations for applicative multiprogramming*, in Lecture Notes in Comput. Sci. 154, Springer-Verlag, New York, 1983, pp. 1–14.

[3] S. Abramsky, *Observation equivalence as testing equivalence*, Theoret. Comput. Sci., 53 (1987), pp. 225–241.

[4] S. Abramsky, *The Lazy Lambda Calculus*, Research Topics in Functional Programming, D. Turner, ed., Addison-Wesley, Reading, MA, 1989, pp. 65–116.

[5] S. Abramsky, *Domain theory in logical form*, Ann. Pure Appl. Logic, 51 (1991), pp. 1–77.

[6] S. Abramsky and C.-H.L. Ong, *Full abstraction in the lazy lambda calculus*, Inform. and Comput., 105 (1993), pp. 159–267.

[7] F. Alessi, *Type preorders*, in Lecture Notes in Comput. Sci. 787, Springer-Verlag, New York, 1994, pp. 37–51.

[8] F. Alessi, M. Dezani-Ciancaglini, and U. de'Liguoro, *May and must convergency in concurrent λ-calculus*, in Lecture Notes in Comput. Sci. 841, Springer-Verlag, New York, 1994, pp. 211–220.

[9] E. A. Ashcroft and M. C. B. Hennessy, *A mathematical semantics for a non-deterministic typed lambda calculus*, Theoret. Comput. Sci., 11 (1980), pp. 227–245.

[10] E. Astesiano and G. Costa, *Non-determinism and fully abstract models*, RAIRO. Inform. Theor. Appl., 14 (1980), pp. 323–347.

[11] E. Astesiano and E. Zucca, *Parametric channels via label expressions in CCS*, Theoret. Comput. Sci., 33 (1984), pp. 45–63.

[12] S. van Bakel, *Complete restrictions of the intersection type discipline*, Theoret. Comput. Sci., 102 (1992), pp. 135–163.

[13] S. van Bakel, *Intersection type assignment systems*, Theoret. Comput. Sci., 151 (1995), pp. 385–436.

[14] F. Barbanera, M. Dezani-Ciancaglini, and U. de'Liguoro, *Intersection and union types: Syntax and semantics*, Inform. and Comput., 119 (1995), pp. 202–230.

[15] H. P. Barendregt, *The Lambda Calculus: Its Syntax and Semantics*, 2nd ed., North-Holland, Amsterdam, 1984.

[16] H. P. Barendregt, M. Coppo, and M. Dezani-Ciancaglini, *A filter lambda model and the completeness of type assignment*, J. Symbol. Logic, 48 (1983), pp. 931–940.

[17] E. W. Beth, *The Foundation of Mathematics*, 2nd ed., North-Holland, Amsterdam, 1965.

[18] G. Boudol, *Semantique Operationelle at Algebrique des Programmes Recursifs Non-Deterministes*, Thèse d'Etat, Université de Paris VII, 1980.

[19] G. Boudol, *A lambda calculus for (strict) parallel functions*, Inform. and Comput., 108 (1994), pp. 51–127.

[20] G. Boudol and C. Laneve, *The discriminating power of multiplicities in the λ-calculus*, Inform. and Comput., 126 (1996), pp. 83–102.

[21] M. Coppo, M. Dezani-Ciancaglini, F. Honsell, and G. Longo, *Extended type structures and filter λ-models*, in Logic Colloquium '82, G. Lolli, G. Longo, and A. Marcja, eds., North-Holland, Amsterdam, 1984, pp. 241–262.

[22] M. Coppo, M. Dezani-Ciancaglini, and G. Longo, *Applicative information systems*, in Lecture Notes in Comput. Sci. 159, Springer-Verlag, New York, 1983, pp. 35–64.

[23] B. A. Davey and H. A. Priestley, *Introduction to Lattices and Order*, Cambridge University Press, Cambridge, UK, 1990.

[24] R. De Nicola and M. Hennessy, *Testing equivalence for proceses*, Theoret. Comput. Sci., 34 (1984), pp. 82–113.

[25] M. Dezani-Ciancaglini, U. de'Liguoro, and A. Piperno, *Filter models for conjunctive-disjunctive λ-calculi*, Theoret. Comput. Sci., 170 (1996), pp. 83–128.

[26] M. Dezani-Ciancaglini, U. de'Liguoro, and A. Piperno, *Fully abstract semantics for concurrent λ-calculus*, Lecture Notes in Comput. Sci. 789, Springer-Verlag, New York, 1994, pp. 16–35.

[27] M. Dezani-Ciancaglini, U. de'Liguoro, and P. van Rossum, *A Type Assignment System for Higher-Order Communicating Processes*, Internal Report, Università di Torino, Italy, 1994.

[28] A. Giacalone, P. Mishra, and S. Prasad, *FACILE: A symmetric integration of concurrent and functional programming*, in Lecture Notes in Comput. Sci. 352, Springer-Verlag, New York, 1989, pp. 184–201.

[29] A. Giacalone, P. Mishra, and S. Prasad, *Operational and algebraic semantics for FACILE: A symmetric integration of concurrent and functional programming*, in Lecture Notes in Comput. Sci. 443, Springer-Verlag, New York, 1990, pp. 765–779.

[30] G. K. Gierz, K. H. Hoffmann, K. Keimel, J. D. Mislove, and D. S. Scott, *A Compendium of Continuous Lattices*, Springer-Verlag, Berlin, 1980.

[31] M. C. B. Hennessy, *The semantics of Call-by-value and call-by-name in a nondeterministic environment*, SIAM J. Comput., 9 (1980), pp. 67–84.

[32] M. C. B. Hennessy, *Powerdomains and non-deterministic recursive definitions*, in Lecture Notes in Comput. Sci. 137, Springer-Verlag, New York, 1982, pp. 178–193.

[33] M. C. B. Hennessy, *Algebraic Theory of Processes*, MIT Press, Cambridge, MA, 1988.

[34] M. C. B. Hennessy, *A fully abstract denotational model for higher-order processes*, Inform. and Comput., 112 (1994), pp. 55–95.

[35] M. C. B. Hennessy, *Higher-order processes and their models*, in Lecture Notes in Comput. Sci. 820, Springer-Verlag, New York, 1994, pp. 286–303.

[36] J. R. Hindley, *The simple semantics for Coppo-Dezani-Sallé type assignment*, in Lecture Notes in Comput. Sci. 137, Springer-Verlag, New York, 1982, pp. 212–226.

[37] J. R. Hindley and G. Longo, *Lambda-calculus models and extensionality*, Z. Math. Logik, 26 (1980), pp. 289–310.

[38] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, Englewood Cliffs, NJ, 1985.

[39] T. Jim and A. R. Meyer, *Full abstraction and the context lemma*, SIAM J. Comput., 25 (1996), pp. 663–696.

[40] P. T. Johnstone, *Stone Spaces*, Cambridge University Press, Cambridge, 1982.

[41] P. J. Landin, *The mechanical evaluation of expressions*, Comput. J., 6 (1964), pp. 308–320.

[42] K. G. Larsen and G. Winskell, *Using information systems to solve recursive domain equations effectively*, in Lecture Notes in Comput. Sci. 173, Springer-Verlag, New York, 1984, pp. 109–130.

[43] C. Lavatelli, *Non-Deterministic Lazy λ-Calculus versus π-Calculus*, LIENS Internal Report 15, Laboratoire d'Informatique de l'Ecole Normale Supérieure, Paris, 1993.

[44] G. Longo, *Set theoretical models of lambda calculus: Theory, expansions and isomorphisms*, Ann. Pure Appl. Logic, 24 (1983), pp. 153–188.

[45] U. de' Liguoro and A. Piperno, *Non-deterministic extensions of untyped λ-calculus*, Inform. and Comput., 122(1995), pp. 149–177.

[46] M. G. Main, *A powerdomain primer*, Bull. EATCS, 33 (1987), pp. 115–147.

[47] J. McCarthy, *A basis for a mathematical theory of computation*, in Computer Programming and Formal Systems, P. Brafford and D. Hirshberg, eds., North-Holland, Amsterdam, 1963, pp. 33–70.

[48] A. Meyer, *What is a model of the lambda calculus?*, Inform. and Comput., 52 (1982), pp. 87–122.

[49] R. Milner, *Fully abstract models of typed λ-calculi*, Theoret. Comput. Sci., 4 (1977), pp. 1–22.

[50] R. Milner, *Function as processes*, J. Math. Struct. Comput. Sci., 2 (1992), pp. 119–141.

[51] R. Milner, J. G. Parrow, and D. J. Walker, *A calculus of mobile processes, Parts* I *and* II, Inform. and Comput., 100 (1992), pp. 1–40, 41–77.

[52] J. H. Morris, *Lambda Calculus Models of Programming Languages*, Ph.d. Thesis, MIT, Cambridge, MA, 1968.

[53] F. Nielson, *The typed λ-calculus with first-class processes*, in Lecture Notes in Comput. Sci. 366, Springer-Verlag, New York, 1989, pp. 357–369.

[54] C.-H.L. Ong, *The Lazy λ-Calculus: An Investigation into the Foundations of Functional Programming*, Ph.D. Thesis, University of London, 1988; also Prize Fellowship Dissertation, Trinity College, Cambridge University, Cambridge, UK.

[55] C.-H.L. Ong, *Concurrent Lambda Calculus and a General Precongruence Theorem for Applicative Bisimulations*, Draft, Cambridge University, Cambridge, UK, 1992.

[56] C.-H.L. Ong, *Non-determinism in a functional setting*, in LICS '93, IEEE Comp. Soc. Press, Los Alamitos, CA, 1993, pp. 275–286.

[57] G. D. Plotkin, *Call-by-name, call-by-value and the λ-calculus*, Theoret. Comput. Sci., 1 (1975), pp. 125–159.

[58] G. D. Plotkin, *A powerdomain construction*, SIAM J. Comput., 5 (1976), pp. 457–487.

[59] G. D. Plotkin, *LCF considered as a programming language*, Theoret. Comput. Sci., 5 (1977), pp. 223–256.

[60] J. H. Reppy, *Concurrent ML: Design, application and semantics*, in Lecture Notes in Comput. Sci. 693, Springer-Verlag, New York, 1993, pp. 165–198.

[61] D. Sangiorgi, *The lazy λ-calculus in a concurrency scenario*, Inform. and Comput., 111 (1994), pp. 120–153.

[62] D. Scott, *A type theoretical alternative to ISWIM, CUCH, OWHY*, Theoret. Comput. Sci., 121 (1993), pp. 411–440.

[63] D. Scott, *Domains for denotational semantics*, in Lecture Notes in Comput. Sci. 140, Springer-Verlag, New York, 1982, pp. 577–613.

[64] K. Sieber, *Call-by-value and non-determinism*, Lecture Notes in Comput. Sci. 664, Springer-Verlag, Berlin, 1993, pp. 376–390.

[65] H. Soendergaard and P. Sestoft, *Non-determinism in functional languages*, Comput. J., 35 (1992), pp. 514–523.

[66] M. B. Smyth, *Power Domains*, J. Comput. System. Sci., 16 (1978), pp. 23–36.

[67] B. Thomsen, *A calculus of higher-order communicating Systems*, in POPL'89, ACM Press, New York, 1989, pp. 142–154.

SIAM J. COMPUT.
Vol. 27, No. 5, pp. 1420–1429, October 1998
© 1998 Society for Industrial and Applied Mathematics
011

DOWNWARD SEPARATION FAILS CATASTROPHICALLY FOR LIMITED NONDETERMINISM CLASSES*

R. BEIGEL† AND J. GOLDSMITH‡

Abstract. The β hierarchy consists of classes $\beta_k = \mathrm{NP}[\log^k n] \subseteq \mathrm{NP}$. Unlike collapses in the polynomial hierarchy and the Boolean hierarchy, collapses in the β hierarchy do not seem to translate up, nor does closure under complement seem to cause the hierarchy to collapse. For any consistent set of collapses and separations of levels of the hierarchy that respects $\mathrm{P} = \beta_1 \subseteq \beta_2 \subseteq \cdots \subseteq \mathrm{NP}$, we can construct an oracle relative to which those collapses and separations hold; at the same time we can make distinct levels of the hierarchy closed under computation or not, as we wish. To give two relatively tame examples: for any $k \geq 1$, we construct an oracle relative to which

$$\mathrm{P} = \beta_k \neq \beta_{k+1} \neq \beta_{k+2} \neq \cdots$$

and another oracle relative to which

$$\mathrm{P} = \beta_k \neq \beta_{k+1} = \mathrm{PSPACE}.$$

We also construct an oracle relative to which $\beta_{2k} = \beta_{2k+1} \neq \beta_{2k+2}$ for all k.

Key words. structural complexity theory, limited nondeterminism, hierarchies, oracles

AMS subject classification. 68Q15

PII. S0097539794277421

1. Introduction.

Although standard nondeterministic algorithms solve many NP-complete problems with $O(n)$ nondeterministic moves, there are other problems that seem to require very different amounts of nondeterminism. For instance, clique can be solved with only $O(\sqrt{n})$ nondeterministic moves, and Pratt's algorithm [16] solves primality, which is not believed to be NP-complete, with $O(n^2)$ nondeterministic moves. Motivated by the different amounts of nondeterminism apparently needed to solve problems in NP, Kintala and Fischer [9, 10, 11] defined limited nondeterminism classes within NP, including the classes we now call the β hierarchy. The structural properties of the β classes were studied further by Àlvarez, Díaz, and Torán [1, 6]. These classes arose yet again in the work of Papadimitriou and Yannakakis [15] on particular problems inside NP (e.g., quasi-group isomorphism can be solved with $O(\log^2 n)$ nondeterministic moves).

Kintala and Fischer [11] defined $\mathcal{P}_{f(n)}$ to be the class of languages accepted by a nondeterministic polynomial-time bounded Turing machine that makes at most $f(n)$ c-ary nondeterministic moves (equivalently, $O(f(n))$ binary nondeterministic moves) on inputs of length n. Being mostly interested in polylogarithmic amounts of nondeterminism, they defined $\mathcal{PL}_k = \mathcal{P}_{\log^k n}$.

Díaz and Torán [6] wrote $\beta_{f(n)}$ to denote Kintala and Fischer's $\mathcal{P}_{f(n)}$, and β_k to denote $\mathcal{PL}_k$. Papadimitriou and Yannakakis [15] wrote $\mathrm{NP}[f(n)]$ to denote $\mathcal{P}_{f(n)}$.

*Received by the editors November 3, 1994; accepted for publication (in revised form) July 31, 1996; published electronically May 19, 1998.
http://www.siam.org/journals/sicomp/27-5/27742.html

†Department of Electrical Engineering and Computer Science, 19 Memorial Drive West, Lehigh University, Packard Laboratory, Bethlehem, PA 18015 (beigel@eecs.lehigh.edu). This research was supported in part by the United States National Science Foundation under grant CCR-8958528 while the author was at Yale University, and by the Netherlands Organization for Scientific Research (NWO) under Visitors Grant B 62-403.

‡University of Kentucky, Department of Computer Science, Lexington, KY 40506 (goldsmit@cs.engr.uky.edu). This research was supported in part by the National Science Foundation under grant CCR-9315354.

(Their work is surveyed in [7].) We will adopt the NP$[f(n)]$ notation of Papadimitriou and Yannakakis, as well as the β_k notation of Díaz and Torán. To reiterate, we offer the following definition.

DEFINITION 1.1.

- *A language L belongs to* NP$[f(n)]$ *if there exists a polynomial-time bounded nondeterministic Turing machine (NTM) that accepts L and makes $O(f(n))$ nondeterministic choices on inputs of length n. (Note.* NP$[f(n)] \subseteq$ DTIME$(2^{O(f(n))})$.*)*
- $\beta_k =$ NP$[\log^k n]$.
- *The β hierarchy consists of β_1, β_2, ...*
- $\beta = \bigcup_k \beta_k$.

Kintala and Fischer [11] constructed oracles that make the β hierarchy collapse to any desired level. That is, there is an oracle relative to which

$$\beta_1 \neq \beta_2 \neq \cdots$$

and, for every $k \geq 1$, there is an oracle relative to which

$$\beta_1 \neq \beta_2 \neq \cdots \neq \beta_k = \text{NP}.$$

Oracles can also make the polynomial hierarchy and the Boolean hierarchy collapse to any desired level [12, 4]. The polynomial and Boolean hierarchies have a very nice property: collapses translate upward. That is, if the kth and $(k+1)$st levels are equal, then all levels are contained in the kth [5, 4]. This is also reflected in the nondeterminism hierarchy, now known as the b hierarchy, studied by Buss and Goldsmith [3]. The classes in the b hierarchy are defined by two parameters: the exponent of the polynomial-time bound (ignoring log factors), and the constant factor for $k \log n$ bits of nondeterminism. This hierarchy exhibits upward collapse for both time and k. All attempts to prove an analogous translational property for the β hierarchy have failed. In fact, the obvious technique extends a collapse by only a constant factor in the number of nondeterministic bits, giving one of the aforementioned upward collapses of the b hierarchy.

Hemachandra and Jha [8] attempted to explain this failure by examining the tally sets in the β hierarchy. For each k, they constructed an oracle that makes $\beta_j \cap \text{TALLY} = \beta_{j+1} \cap \text{TALLY}$ if and only if $j \leq k$. We find this explanation unsatisfactory because it considers only tally sets.

The known behavior of relativized β hierarchies is that $\beta_0^A = \beta_1^A$ and that $\beta_i^A \subseteq \beta_j^A$ for all $i < j$ and all oracles A. A *collapse* is a statement of the form $\beta_i^A = \beta_j^A$. A *separation* is a statement of the form $\beta_i^A \neq \beta_j^A$. A *closure* is a statement of the form $\beta_i^A =$ co-β_i^A. A *nonclosure* is a statement of the form $\beta_i^A \neq$ co-β_i^A. A *requirement* is a collapse, separation, closure, or nonclosure. We call a set of requirements *consistent* if it is consistent with the known behavior of relativized β hierarchies, as stated above, and the standard axioms for $=$ and $\subseteq$.

Given a set S of requirements, let X be the union of $[0,1]$ and all intervals $[i,j]$ such that $i < j$ and the collapse $\beta_i^A = \beta_j^A$ belongs to S. It is easy to see that S is consistent if and only if the following conditions hold for all a and b such that $[a,b] \subseteq X$ or $[b,a] \subseteq X$: (1) the separation $\beta_a^A \neq \beta_b^A$ does not belong to S; (2) the closure $\beta_a^A =$ co-β_a^A and the nonclosure $\beta_b^A \neq$ co-β_b^A do not both belong to S.

For any consistent set of requirements, we construct an oracle A such that the β hierarchy relative to A satisfies them. For example, for each $k \geq 0$, there is an oracle that makes $\beta_j = \beta_{j+1}$ if and only if $j \leq k$. Another oracle makes $\beta_j = \beta_{j+1}$ if

and only if $j = 0$ or j is prime. We can also make distinct levels in the hierarchy be closed under complementation or not, as long as this is consistent with the collapses (if $\beta_i^A = \beta_j^A$, then we cannot have $\beta_i^A = \text{co-}\beta_i^A$ and $\beta_j^A \neq \text{co-}\beta_j^A$).

We prove two initial results for every k.

- *There is an oracle that makes the first k levels of the β hierarchy coincide, but makes the remaining levels all distinct (Theorem* 2.3*).*
- *There is an oracle that makes the first k levels of the β hierarchy coincide, the $(k+1)$st level different from the kth, and the remaining levels all equal (Theorem* 2.5*).*

The techniques from these two constructions can be combined to get any consistent finite or infinite set of collapses and separations. To collapse β_k into β_j (for $k > j$), we code a complete set for β_k into β_j. The same coding techniques can also code co-$\overline{\beta}_i$ into β_i, for any i's we wish, as long as this doesn't violate any collapses. (If β_k is collapsed to β_j, then either both or neither will be closed under complement.) Finally, in section 3, we extend our results to β_r for real $r \geq 0$.

One theme in complexity theory is to ask whether $\text{NP} - \text{P}$ contains any easy sets (assuming $\text{P} \neq \text{NP}$). The answer to the question above depends on the definition of "easy." Ladner [14] showed that if $\text{P} \neq \text{NP}$ then $\text{NP} - \text{P}$ contains an incomplete set. On the other hand, there are oracles relative to which $\text{P} \neq \text{NP}$, but $\text{NP} - \text{P}$ contains (a) no tally sets [13] or (b) no sets in co-NP [2]. It is unknown whether the assumption $\text{P} \neq \text{NP}$ implies that $\text{NP} - \text{P}$ contains a set in $\text{DTIME}(n^{\text{polylog}})$; a positive answer would improve many constructions in the literature. As a step toward understanding that question, we construct an oracle relative to which $\text{P} \neq \text{NP}$ but $\text{NP} - \text{P}$ contains no set in the β hierarchy (Corollary 2.2).

2. Limited nondeterminism hierarchies. The construction below gives almost all the techniques used in subsequent theorems.

THEOREM 2.1. *Let g_0 and g_1 be polynomial-time computable monotone increasing functions such that $\log n \in o(g_1(n))$ and $g_0(n) \in n^{O(1)}$. If $g_0(n^{O(1)}) \subseteq o(g_1(n))$, then there exists an oracle A such that $\text{P}^A = (\text{NP}[g_0(n)])^A \neq (\text{NP}[g_1(n)])^A$ (and in fact there is a tally set in $(\text{NP}[g_1(n)])^A - (\text{NP}[g_0(n)])^A$).*

Proof. Let $C^A = \{(e, x, 0^s) : \text{oracle NTM } e \text{ accepts } x \text{ within } s \text{ steps with oracle } A, \text{ making at most } g_0(|x|) \text{ nondeterministic choices}\}$. Then C^A is $\leq_m^p$-complete for $(\text{NP}[g_0(n)])^A$ for every A. Let $p(n)$ be the polynomial-time bound for some $\text{NP}[g_0(n)]$ oracle Turing machine recognizing $C^{()}$.

Let $D^A = \{x \in 1^* : \exists y[|y| = g_1(|x|) \wedge 0xy \in A]\}$. Note that $D^A \in \text{NP}[g_1(n)]^A$.

The construction consists of coding C^A into A in a polynomial-time recoverable manner, making $(\text{NP}[g_0(n)])^A \subseteq \text{P}^A$, while diagonalizing, i.e., guaranteeing that no P^A machine recognizes the set D^A, so $\text{P}^A \neq (\text{NP}[g_1(n)])^A$.

At the end of the construction, we will have

$$w \in C^A \iff 1^{p(|w|)^2} 0w \in A.$$

We refer to all strings beginning with 1 as *coding strings.* We refer to all strings beginning with 0 as *diagonalizing strings.*

Assume that $\text{P}^{()}$ is enumerated by deterministic Turing machines (DTMs) $P_1^{()}$, $P_2^{()}, \ldots$ so that for all i, $P_i^{()}$ runs in time bounded by n^i for sufficiently large n.

The construction proceeds in stages. At the end of stage s, A is decided for all strings of length up to n_s, and D^A is extended so that P_s^A does not recognize D^A. The stage consists of one diagonalization, which determines n_s, and continued encoding of C^A.

At stage s, choose $n > n_{s-1}$ such that n is a power of 2, the running time of $P_s^{()}$ on inputs of length n is at most n^s, and n satisfies an inequality that is to be specified below and that is true for almost all n. Let $x = 1^n$. The value of $D^A(1^n)$ depends only on oracle strings of length $\ell = 1 + n + g_1(n)$. Code C^A for all coding strings of length $\leq \ell - 1$ (i.e., for all w such that $p(|w|)^2 + 1 + |w| \leq \ell - 1$, let $1^{p(|w|)^2}0w \in A \iff w \in C^A$) and freeze A up to length $\ell - 1$.

In order to diagonalize, $P_s^A(x)$ must be calculated. But that computation may query coding strings that code computations of C^A that are not yet decided, because those computations in turn rely on strings for which A is not yet decided. Those strings in turn may depend on other coding strings. Any diagonalizing strings that do not already belong to A and are queried by $P_s^A(x)$, or by the computation corresponding to a coding string that $P_s^A(x)$ queries, or in the computation corresponding to a coding string that one of those computations queries, or so on, are restrained from A. We claim that there are more potential witnesses for x to be in D^A than there are possible queries in such a cascade of queries, so deciding $P_s^A(x)$ does not restrict our decision about $D^A(x)$.

Because of the encoding of C^A, a coding string z codes a computation that depends only on strings of length bounded by $\sqrt{|z|}$. $C^A(w)$ directly depends on at most $p(|w|)2^{g_0(|w|)}$ of these shorter strings.

A computation of $P_s^A(x)$ may query no more than n^s strings, each of length bounded by n^s. Each of these strings may code a computation on a string of length at most $n^{s/2}$. Each of these computations depends on at most $p(n^{s/2})2^{g_0(n^{s/2})}$ strings, each of which depends on at most $p(n^{s/4})2^{g_0(n^{s/4})}$ strings, etc.

This recursion can be cut off at strings of length $\ell - 1$, because A is already determined up to that length. The total number of queries needed to decide $P_s^A(x)$ is bounded by n^s times the product of all the terms above of the form $p(n^{s/2^i})2^{g_0(n^{s/2^i})}$. There are at most $\log\log n^s - \log\log(\ell - 1) \leq \log\log n^s - \log\log n = \log s$ such terms, and each of them is bounded by $p(n^s)2^{g_0(n^s)} = 2^{O(g_0(n^s))}$. Therefore, the total number of queries on which $P_s^A(x)$ depends is $n^s 2^{O(\log(s) g_0(n^s))}$, which is less than $2^{g_1(n)}$ for sufficiently large n. (The inequality that n must satisfy is $n^s(p(n^s)2^{g_0(n^s)})^{\log(s)} < 2^{g_1(n)}$.)

Thus there remains an unrestrained diagonalizing string of length ℓ, which we put into A if $P_s^A(x)$ rejects x. That is, we set $D^A(x) = 1 - P_s^A(x)$, adding a string $0xy$ to A if necessary. Thus, for each s, we can guarantee that P_s^A does not accept D^A, so $D^A \notin \mathrm{P}^A$. Since $D^A \in (\mathrm{NP}[g_1(n)])^A$, this shows that $(\mathrm{NP}[g_1(n)])^A \neq \mathrm{P}^A$. Since $C^A \in \mathrm{P}^A$ is complete for $(\mathrm{NP}[g_0(n)])^A$, this shows that $(\mathrm{NP}[g_0(n)])^A \subseteq \mathrm{P}^A$.

We complete stage s by letting $n_s = \max(\ell, n^s)$ and finishing the coding of any $C^A(w)$ that was begun or queried in this stage. □

The preceding theorem is tight because if $\mathrm{P} = \mathrm{NP}[g_0(n)]$ then $\mathrm{P} = \mathrm{NP}[g_0(n^{O(1)})]$ (even if we restrict to binary nondeterministic moves) via a relativizable proof. (Previously, Sanchis [17] had observed that if $\mathrm{P} = \mathrm{NP}[n^\epsilon]$ for some $\epsilon > 0$, then $\mathrm{P} = \mathrm{NP}$.)

Because the classes are separated by tally sets, we also separate the exponential-time versions of these classes (see [8] for an elaboration of this).

COROLLARY 2.2. *There is an oracle relative to which* $\mathrm{P} = \beta \neq \mathrm{NP}$.

Proof. Let $g_0(n) = 2^{(\log\log n)^2}$ and $g_1(n) = n$ in the previous theorem. Then, for all k, $\beta_k^A \subseteq (\mathrm{NP}[2^{(\log\log n)^2}])^A = \mathrm{P}^A \neq \mathrm{NP}^A$. □

THEOREM 2.3. *Let $g(\cdot,\cdot)$ be a polynomial-time computable, monotone increasing (in both variables) function with $\log n \in o(g(n,i))$ and $g(n,i) \in n^{O(1)}$ for all $i \geq 0$. If $g(n^{O(1)}, 2i) \subseteq o(g(n, 2i+1))$ for all $i \geq 1$, then there exists an oracle A such that*

$\mathrm{P}^A = (\mathrm{NP}[g(n,0)])^A$ *and* $(\mathrm{NP}[g(n,2i)])^A \neq (\mathrm{NP}[g(n,2i+1)])^A$ *for all* $i \geq 0$ *(and in fact there is a tally set in* $(\mathrm{NP}[g(n,2i+1)])^A - (\mathrm{NP}[g(n,2i)])^A$ *for each* i*).*

(In this theorem, we ignore the relationship between $\mathrm{NP}[g(n,2i-1)]$ and $\mathrm{NP}[g(n,2i)]$. We will take that up in the next theorem.) The only difference between this construction and the previous one is that there are infinitely many diagonalizations going on. At stage $s = \langle e, i\rangle$, we guarantee that the eth machine for $(\mathrm{NP}[g(n,2i)])^A$ does not accept the diagonal set $D_i^A \in (\mathrm{NP}[g(n,2i+1)])^A$. Thus, $(\mathrm{NP}[g(n,2i+1)])^A \neq (\mathrm{NP}[g(n,2i)])^A$. The counting argument for this construction is identical to that in the proof of Theorem 2.1.

COROLLARY 2.4. *For any* k*, there is an oracle relative to which*

$$\mathrm{P} = \beta_k \neq \beta_{k+1} \neq \beta_{k+2} \neq \cdots.$$

Proof. Let $g(n,i) = \log^{k+\lfloor i/2\rfloor} n$ in the preceding theorem. □

THEOREM 2.5. *Let* g_0 *and* g_1 *be polynomial-time computable, monotone increasing functions such that* $\log n \in o(g_1(n))$ *and* $g_0(n) \in n^{O(1)}$*. If* $g_0(n^{O(1)}) \subseteq o(g_1(n))$ *then there exists an oracle* A *such that*

$$\mathrm{P}^A = (\mathrm{NP}[g_0(n)])^A \neq (\mathrm{NP}[g_1(n)])^A = \mathrm{PSPACE}^A$$

(and in fact there is a tally set in $(\mathrm{NP}[g_1(n)])^A - (\mathrm{NP}[g_0(n)])^A$*).*

Sketch. In this construction, we do two encodings and one diagonalization. In addition to coding C_k^A into P, we also code E^A, a generic $\leq_m^p$-complete set for PSPACE, into A. ($E^A = \{(e,x,0^s) :$ oracle DTM e accepts x using at most s tape squares with oracle $A\}$, where we also count the space used on the oracle tape.) At the end of the construction, we have

$$x \in E^A \iff (\exists y)[|y| = g_1(n) \wedge 2xy \in A].$$

(If one prefers binary oracles, one may code 0, 1, and 2 as 00, 01, and 10.) When we are doing a diagonalization to make $P_s^A(x) \neq D^A(x)$, if a coding string for $C_k^A(w)$ is queried, we proceed as before; if a coding string for $E^A(w)$ is queried, where $|w| \geq |x|$, then we simply restrain that coding string from the oracle. This will not restrain all the coding strings for $E^A(w)$, since there are $2^{g_1(|w|)}$ coding strings for $E^A(w)$; if $|w| \geq |x|$, then $2^{g_1(|w|)} > (p(|x|^s)2^{g_0(|x|^s)})^{\log(s)}$, where $(p(|x|^s)2^{g_0(|x|^s)})^{\log(s)}$ is the upper bound on the total number of queries generated by the computation of $P_s^A(x)$, as in the proof of Theorem 2.1. Therefore, restraining any such coding strings queried in the computation of $P_s^A(x)$ or in its cascade of queries cannot restrain all such coding strings, and thus cannot decide $E^A(w)$. At the end of each stage, we complete all codings begun or queried in that stage, so that it will not be changed in any subsequent stage. □

COROLLARY 2.6. *For every* k*, there is an oracle relative to which*

$$\mathrm{P} = \beta_k \neq \beta_{k+1} = \mathrm{PSPACE}.$$

With only a slight modification of this technique, we get far more bizarre collapses.

THEOREM 2.7. *Let* $g(\cdot,\cdot)$ *be a polynomial-time computable, monotone increasing (in both variables) function with* $\log n \in o(g(n,i))$ *and* $g(n,i) \in n^{O(1)}$ *for all* $i \geq 1$*. If* $g(n^{O(1)}, 2i) \subseteq o(g(n,2i+1))$ *for all* $i \geq 1$*, then there exists an oracle* A *such that*

$\mathrm{P}^A = (\mathrm{NP}[g(n,0)])^A$ *and* $(\mathrm{NP}[g(n,2i)])^A \neq (\mathrm{NP}[g(n,2i+1)])^A = (\mathrm{NP}[g(n,2i+2)])^A$ *for all* $i \geq 0$ *(and in fact there is a tally set in* $(\mathrm{NP}[g(n,2i+1)])^A - (\mathrm{NP}[g(n,2i)])^A$ *for each* i*).*

We include the full proof of this result, although it uses techniques mentioned before, since this shows how all the pieces fit together.

Proof. Let $C_i^A = \{(e,x,0^s) :$ oracle NTM e accepts x within s steps with oracle A, making at most $g(|x|,i)$ nondeterministic choices$\}$. Then C_i^A is $\leq_m^p$-complete for $(\mathrm{NP}[g(n,i)])^A$ for any A. Let $p(n,i)$ be the nondeterministic time bound for some $\mathrm{NP}[g(n,i)]$ oracle Turing machine recognizing $C_i^{()}$. Without loss of generality, assume that for all i and almost all n, $p(n,i) \leq p(n,i+1)$.

Let $D_i^A = \{x : \exists y[|y| = g(|x|,i) \wedge 0xy \in A]\}$.

For convenience, define $g(n,-1) = 0$ for all n.

The construction consists of coding C_{2i}^A into $(\mathrm{NP}[g(n,2i-1)])^A$, for each $i \geq 0$, so $(\mathrm{NP}[g(n,0)])^A \subseteq \mathrm{P}^A$ and $(\mathrm{NP}[g(n,2i+2)])^A \subseteq (\mathrm{NP}[g(n,2i+1)])^A$ for all i, and diagonalizing, i.e., guaranteeing that no $(\mathrm{NP}[g(n,2i)])^A$ machine recognizes the set D_{2i+1}^A, for any i, so $(\mathrm{NP}[g(n,2i+1)])^A \not\subseteq (\mathrm{NP}[g(n,2i)])^A$ for any i.

At the end of the construction, we will have $x \in C_{2i}^A \iff \exists y[|y| = g(|x|,2i-1) \wedge 10^{2i}1^{p(|x|,2i-1)^2}0xy \in A]$.

Assume that $(\mathrm{NP}[g(n,i)])^A$ is enumerated by oracle NTMs $M_{1,i}, M_{2,i}, \ldots$, where $M_{e,i}$ runs in time bounded by n^e for sufficiently large n.

The construction proceeds in stages. Stage $s = \langle e,i\rangle$ consists of some encodings and one diagonalization, which determines n_s. At the end of stage s, A is decided for all strings of length $\leq n_s$ (and some further coding strings), and A has been extended so that $M_{e,2i}^A$ does not recognize D_{2i+1}^A.

At stage s, let $\langle e,i\rangle = s$, and then choose $n > n_{s-1}$ such that n is a power of 2, $M_{e,2i}$ runs in time bounded by n^e on inputs of length n, and n satisfies an inequality to be specified below that is true for almost all n. Let $n_s = n$. Let $x = 0^n$. The value of $D_{2i+1}^A(x)$ depends only on strings of length $\ell = 1 + n + g(n,2i+1)$. Do all coding involving witnesses of length less than ℓ, and then freeze A through length $\ell - 1$.

As before, in order to diagonalize we will need to calculate $M_{e,2i}^A(x)$ which may generate a cascade of queries. Any diagonalizing strings that do not already belong to A and are queried in this cascade are restrained from A. But coding strings may be queried as well. (Because we are coding nondeterministically, coding strings can be thought of as potential witnesses to membership.) If $M_{e,2i}^A(x)$ queries a potential witness that $w \in C_{2j}^A$, where $2j > 2i$ and $C_{2j}^A(w)$ has not yet been decided, that potential witness is restrained from A. If $2j \leq 2i$ and $C_{2j}^A(w)$ has not yet been decided, then we compute $C_{2j}^A(w)$ recursively. We will show below that the number of queries generated by such a cascade of queries is smaller than both of the following bounds: (1) the number of potential witnesses for $w \in C_{2j}^A$, (2) the number of potential witnesses for $x \in D_{2i+1}^A$. In fact, bound (1) implies bound (2) as follows. The number of witnesses for $D_{2i+1}^A(x)$ is $2^{g(|x|,2i+1)}$, and the number of witnesses for $C_{2j}^A(w)$ is $2^{g(|w|,2j-1)}$. If a witness of $C_{2j}^A(w)$ is restrained, then $|w| \geq |x|$ and $2j > 2i$. Thus $2j-1 \geq 2i+1$, so, by monotonicity of g, $g(|w|,2j-1) \geq g(|x|,2j-1) \geq g(|x|,2i+1)$.

Thus, restraining potential witnesses as described does not impede any encodings or restrict our decisions about $D_{2i+1}(x)$ or those $C_{2j}^A(w)$ for which we restrict coding strings. (We don't have to worry about what happens to potential witnesses for $w \in C_{2j}^A$ at a later stage, because any affected codings, e.g., $C_{2j}^A(w)$, will be completed at this stage; later diagonalizations will not affect them.)

Now we show that there are more potential witnesses for $x \in D^A_{2i+1}$ than there are possible queries in such a cascade of queries. Because of how we encode C^A_k, a coding string z codes a computation that depends on strings of length bounded by $\sqrt{|z|}$. For $k \le 2i+1$, $C^A_k(w)$ depends on at most $p(|w|,k)2^{g(|w|,k)} \le p(|w|,2i+1)2^{g(|w|,2i+1)}$ of these shorter strings.

$M^A_{e,2i}(x)$ has at most $2^{g(n,2i)}$ computations, and each of those computations may query no more than n^e strings, each of length bounded by n^e. Each such string may code a computation $C^A_{2j}(w)$ where $|w| \le \sqrt{n^e}$, but we only need to expand that computation if $2j \le 2i$. Each of these computations depends on at most $p(n^{e/2},2i)2^{g(n^{e/2},2i)}$ strings, each of which depends on at most $p(n^{e/4},2i)2^{g(n^{e/4},2i)}$ strings, etc. As before, the total number of queries needed to decide $M^A_{e,i}(x)$ is bounded by the product of $\log e \le \log s$ terms, each of which is $2^{o(g(n,2i+1))}$. Therefore the total number of queries on which $M^A_{e,2i}(x)$ depends is $2^{o(g(n,2i+1))}$, which is less than $2^{g(n,2i+1)}$ for sufficiently large n.

Thus there remains an unrestrained diagonalizing string of length ℓ, which we put into A if and only if $M^A_{e,2i}(x)$ rejects x. That is, we set $D^A_{2i+1}(x) = 1 - M^A_{e,2i}(x)$, adding a string $0xy$ to A if necessary. Thus, for each s, we can guarantee that $M^A_{e,2i}$ does not accept D^A_{2i+1}, so $D^A_{2i+1} \notin (\mathrm{NP}[g(n,2i)])^A$. Since $D^A_i \in (\mathrm{NP}[g(n,2i+1)])^A$, this shows that $(\mathrm{NP}[g(n,2i+1)])^A \not\subseteq (\mathrm{NP}[g(n,2i)])^A$ for all $i \ge 0$.

Since C^A_{2i} is complete for $(\mathrm{NP}[g(n,2i)])^A$, our encoding guarantees that $(\mathrm{NP}[g(n,2i)])^A \subseteq (\mathrm{NP}[g(n,2i-1)])^A$ for all $i \ge 0$. □

Corollary 2.8. *There is an oracle relative to which, for each k, $\beta_{2k} = \beta_{2k+1} \neq \beta_{2k+2}$.*

Corollary 2.9. *For any consistent pattern of collapses and separations of the β_k's, there is an oracle relative to which that pattern holds.*

Notice that if the set of collapses is not recursive, then the oracle will also be nonrecursive.

In addition to collapsing or separating β_j and β_k, we can code co-β_k into β_k—or separate the two. This involves some additional argument.

Theorem 2.10. *Let $g(\cdot,\cdot)$ be a polynomial-time computable, monotone increasing (in both variables) function with $\log n \in o(g(n,i))$ and $g(n,i) \in n^{O(1)}$ for all $i \ge 1$. If $g(n^{O(1)},2i) \subseteq o(g(n,2i+1))$ for all $i \ge 1$, then there exists an oracle A such that $\mathrm{P}^A = (\mathrm{NP}[g(n,0)])^A$ and $(\mathrm{NP}[g(n,2i)])^A \neq (\mathrm{NP}[g(n,2i+1)])^A = (\mathrm{NP}[g(n,2i+2)])^A \neq (\text{co-NP}[g(n,2i+2)])^A$ for all $i \ge 0$ (and in fact there are tally sets in $(\mathrm{NP}[g(n,2i+1)])^A - (\mathrm{NP}[g(n,2i)])^A$ and in $(\mathrm{NP}[g(n,2i+2)])^A - (\text{co-NP}[g(n,2i+2)])^A$ for each i).*

Sketch. We will separate $(\mathrm{NP}[g(n,2i+1)])^A$ from $(\text{co-NP}[g(n,2i+1)])^A$ rather than $(\mathrm{NP}[g(n,2i+2)])^A$ from $(\text{co-NP}[g(n,2i+2)])^A$; given the other requirements, this is equivalent. In order to separate $(\mathrm{NP}[g(n,2i+1)])^A$ from $(\text{co-NP}[g(n,2i+1)])^A$, we use the set D^A_{2i+1} in $(\mathrm{NP}[g(n,2i+1)])^A$ and guarantee that $D^A_{2i+1} \notin (\text{co-NP}[g(n,2i+1)])^A$. Most of this construction is identical to that of Theorem 2.7, except that we interleave an extra diagonalization into the construction; the codings and diagonalizations are analogous to earlier constructions, and the counting argument is identical.

We code complete sets C^A_{2i} for $(\mathrm{NP}[g(n,2i)])^A$ into $(\mathrm{NP}[g(n,2i-1)])^A$, and diagonalize so that no $(\mathrm{NP}[g(n,2i)])^A$ machine recognizes $D^A_{2i+1} \in (\mathrm{NP}[g(n,2i+1)])^A$. (Thus D^A_{2i+1} does double duty: during even stages, it diagonalizes against $(\mathrm{NP}[g(n,2i+2)])^A$, and during odd stages, against $(\text{co-NP}[g(n,2i+1)])^A$.)

To guarantee that D^A_{2i+1} is not in $(\text{co-NP}[g(n,2i+1)])^A$, we make sure that, for

each e, the eth machine for $(\mathrm{NP}[g(n,2i+1)])^A$ does not recognize $\overline{D^A_{2i+1}}$. This holds if and only if there is some x such that $D^A_{2i+1}(x) = M^A_{e,2i+1}(x)$. This diagonalization differs from earlier ones only when $M^A_{e,2i+1}(x)$ queries a witness for $x \in D^A_{2i+1}$. As before, if $M^A_{e,2i+1}(x)$ queries a coding string for some computation of $C^A_{2j}(w)$ where $2j > 2i$ or $2j = 2i$ and $w \neq x$, then we can safely restrain the coding string. (If $2j = 2i$ and $w \neq x$, this may exclude w from D^A_{2i+1}, but that doesn't matter. As long as $D^A_{2i+1}(x) = M^A_{e,2i+1}(x)$, we don't care what happens to D^A_{2i+1} for other strings of lengths between n_{s-1} and n_s, where $s = \langle e, i\rangle$.) If $2j < 2i$, then we retrace the computation, as before.

If $M^A_{e,2i+1}(x)$ queries a witness for $x \in D^A_{2i+1}$, we first restrain all such witnesses and continue. If this leads to a rejecting computation of $M^A_{e,2i+1}(x)$, then $M^A_{e,2i+1}(x) = D^A_{2i+1}(x)$, and the diagonalization is successful. If it leads to an accepting computation, we preserve the lexicographically least accepting path for that computation, and all of its cascade of queries. As before, the computation of $M^A_{e,2i+1}(x)$ restrains at most $2^{o(g(n,2i+1))}$ strings, so this will not restrain all the witnesses for $x \in D^A_{2i+1}$. Thus we can find an unrestrained witness and add it to A, so $D^A_{2i+1}(x) = M^A_{e,2i+1}(x)$, as desired.

Therefore, this additional set of diagonalization requirements can be interleaved with the previously described diagonalizations and collapses. □

THEOREM 2.11. *Let $g(\cdot,\cdot)$ be a polynomial-time computable, monotone increasing (in both variables) function with $\log n \in o(g(n,i))$ and $g(n,i) \in n^{O(1)}$ for all $i \geq 1$. If $g(n^{O(1)}, 2i) \subseteq o(g(n, 2i+1))$ for all $i \geq 1$, then there exists an oracle A such that $\mathrm{P}^A = (\mathrm{NP}[g(n,0)])^A$ and $(\mathrm{NP}[g(n,2i)])^A \neq (\mathrm{NP}[g(n,2i+1)])^A = (\mathrm{NP}[g(n,2i+2)])^A = (\text{co-NP}[g(n,2i+2)])^A$ for all $i \geq 0$ (and in fact there are tally sets in $(\mathrm{NP}[g(n,2i+1)])^A - (\mathrm{NP}[g(n,2i)])^A$ for each i).*

Sketch. As before, we construct A so that no $(\mathrm{NP}[g(n,2i)])^A$ machine recognizes the set D^A_{2i+1}, and so that $C^A_{2i+2} \in (\mathrm{NP}[g(n,2i+1)])^A$. In addition, in order to make $(\mathrm{NP}[g(n,2i+1)])^A = (\text{co-NP}[g(n,2i+1)])^A$, we code $\overline{C^A_{2i+1}}$ into $(\mathrm{NP}[g(n,2i+1)])^A$ as follows: $x \notin C^A_{2i+1} \iff \exists y[|y| = g(|x|, 2i+1) \wedge 21^{p(|x|,2i+1)^2}0x \in A]$.

For each i, let N^A_i be an $(\mathrm{NP}[g(n,i)])^A$ machine recognizing C^A_i in nondeterministic time bounded by $p(n,i)$ (regardless of the oracle). By the form of the encoding, $N^A_i(x)$ cannot query any of its own coding strings. If a witness string for $\overline{C^A_{2j+1}(x)}$ is queried in the course of a diagonalization $(\mathrm{NP}[g(n,2i)])^A \neq (\mathrm{NP}[g(n,2i+1)])^A$, and $2j+1 < 2i+1$, then we can retrace the computation. If $2j+1 \geq 2i+1$, then we can restrain the queried witness string (for $|x|$ sufficiently large) without deciding $\overline{C^A_{2j+1}(x)}$, by the same counting argument as in previous proofs.

Thus, we can add this extra encoding, without interfering with the other collapses and codings. □

This gives us the following stronger version of Hemachandra and Jha's oracle [8].

COROLLARY 2.12. *For each k, there is an oracle relative to which for all j, $\beta_j = \text{co-}\beta_j$, and*

$$\mathrm{P} = \beta_k \neq \beta_{k+1} \neq \beta_{k+2} \neq \cdots$$

(and the separations are witnessed by tally sets).

Combining the results (and techniques) of Theorems 2.7, 2.10, and 2.11, we get the following very strong result.

COROLLARY 2.13. *For any consistent set of requirements, there is an oracle relative to which the β hierarchy satisfies those requirements.*

In constructing such an oracle, one must be careful in closing classes under complement. In particular, if we close one class under complement and separate another from its complement, we cannot then make the two classes equal.

Corollary 2.9 implies that there are uncountably many different patterns of collapse that can be realized in relativized worlds. If the set of requirements is recursive, then the oracle can be made recursive, but certainly some of those patterns are realized by only nonrecursive oracles.

3. Dense β hierarchies. Previously we considered β_r only when r is a natural number. But the class $(\mathrm{NP}[\log^r n])^A$ is meaningful whenever r is a nonnegative real number (regardless of whether r is computable). Even when we allow r to be real, we can make the β hierarchy obey any consistent set of requirements. For example, we can make the β hierarchy look like a Cantor set.

THEOREM 3.1. *Let X be any subset of $[1,\infty)$. There exists an oracle A such that, for all $s,t \geq 1$, $\beta_s^A = \beta_t^A$ if and only if $[s,t] \subseteq X$.*

Note that there may be uncountably many distinct β_t's. Because there are two rationals between any two reals, we need only separate the distinct β_q's where q is rational.

Proof. Without loss of generality, assume that X is a union of intervals, each containing more than one point. Every interval in X contains a rational point; therefore X contains countably many intervals.

We will satisfy the following requirements for each maximal interval in X, depending on its type:
$[s,t]$: $\mathrm{NP}[\log^s n] = \mathrm{NP}[\log^t n]$,
$[s,t)$: $\mathrm{NP}[\log^s n] = \mathrm{NP}[\log^t n/\log\log n]$,
$(s,t]$: $\mathrm{NP}[\log^s n \log\log n] = \mathrm{NP}[\log^t n]$,
(s,t) : $\mathrm{NP}[\log^s n \log\log n] = \mathrm{NP}[\log^t n/\log\log n]$,
$[s,\infty)$: $\mathrm{NP}[\log^s n] = \mathrm{NP}[n]$,
(s,∞) : $\mathrm{NP}[\log^s n \log\log n] = \mathrm{NP}[n]$.
In addition, for each rational number q in $(1,\infty) - X$, we will make

$$\mathrm{NP}[\log^q n/\log\log n] \neq \mathrm{NP}[\log^q n] \neq \mathrm{NP}[\log^q n \log\log n].$$

If $1 \notin X$, then we make $\mathrm{P} \neq \mathrm{NP}[\log n \log\log n]$ as well.

The construction is a slight modification of that in the proof of Theorem 2.7. We perform the diagonalizations in some well-founded order, while maintaining the codings as we go along. The only significant difference here is that the diagonalizations are not performed in increasing order. Suppose that at some stage we are making $\mathrm{NP}[a(n)] \subset \mathrm{NP}[b(n)]$ and a coding string for some $\mathrm{NP}[c(n)]$ computation is queried; we restrain that string if and only if $(\exists n)[c(n) > b(n)]$ if and only if $(\forall n)[c(n) \geq b(n)]$. The counting argument is the same as before. □

Note that we could also close each distinct β_r under complement or not, as we wish, in the theorem above.

4. Open problems. The class β_k is contained in $\mathrm{NP} \cap \mathrm{DTIME}(2^{\log^k n})$. Our work shows that there is no relativizing proof that $\mathrm{P} = \beta_2 \Rightarrow \mathrm{P} = \mathrm{NP}$. We would like to know whether $\mathrm{P} = \mathrm{NP} \cap \mathrm{DTIME}(n^{\log n})$ implies $\mathrm{P} = \mathrm{NP}$; i.e., if $\mathrm{P} \neq \mathrm{NP}$, are there any easy languages in $\mathrm{NP} - \mathrm{P}$? The best we can show is that if $\mathrm{P} = \mathrm{NP} \cap \mathrm{DTIME}(f(n))$ for a well-behaved function f, then $\mathrm{P} = \mathrm{NP} \cap \mathrm{DTIME}(f(f(n)))$. Is there an oracle relative to which this is the best possible translation of the collapse? Does $\mathrm{P} = \mathrm{NP} \cap \mathrm{DTIME}(2^{\log^k n})$ imply $\mathrm{P} = \mathrm{NP}$?

Acknowledgments. We are grateful to Leen Torenvliet, Andrew Klapper, and Martin Kummer for helpful discussions, and Andrew Klapper, Bill Gasarch, and Martin Kummer for careful proofreading of earlier drafts.

REFERENCES

[1] C. Àlvarez, J. Díaz, and J. Torán, *Complexity Classes With Complete Problems Between P and* NP*-Complete*, in Foundations of Computation Theory, Lecture Notes in Computer Science 380, Springer-Verlag, New York, 1989, pp. 13–24.

[2] T. Baker, J. Gill, and R. Solovay, *Relativizations of the* P = NP *question*, SIAM J. Comput., 4 (1975), pp. 431–442.

[3] J. F. Buss and J. Goldsmith, *Nondeterminism within P*, SIAM J. Comput., 22 (1993), pp. 560–572.

[4] J. Cai, T. Gundermann, J. Hartmanis, L. A. Hemachandra, V. Sewelson, K. W. Wagner, and G. Wechsung, *The Boolean hierarchy* I*: Structural properties*, SIAM J. Comput., 17 (1988), pp. 1232–1252.

[5] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer, *Alternation*, J. Assoc. Comput. Mach., 28 (1981), pp. 114–133.

[6] J. Díaz and J. Torán, *Classes of bounded nondeterminism*, Math. Systems Theory, 23 (1990), pp. 21–32.

[7] J. Goldsmith, M. Levy, and M. Mundhenk, *Limited nondeterminism*, SIGACT News, June 1996, pp. 20–29.

[8] L. Hemachandra and S. Jha, *Defying upward and downward separation*, Inform. and Comput., 121 (1995), pp. 1–13.

[9] C. M. R. Kintala, *Computations with a Restricted Number of Nondeterministic Steps*, Ph.D. Thesis, Pennsylvania State University, University Park, PA, 1977.

[10] C. M. R. Kintala and P. C. Fischer, *Computations with a restricted number of nondeterministic steps*, in 9th Ann. ACM Symp. on Theory of Comput., ACM, New York, 1977, pp. 178–185.

[11] C. M. R. Kintala and P. C. Fischer, *Refining nondeterminism in relativized polynomial-time bounded computations*, SIAM J. Comput., 9 (1980), pp. 46–53.

[12] K.-I. Ko, *Relativized polynomial hierarchies having exactly k levels*, SIAM J. Comput., 18 (1989), pp. 392–408.

[13] S. Kurtz, *Sparse sets in* NP − P*: Relativizations*, SIAM J. Comput., 14 (1985), pp. 113–119.

[14] R. E. Ladner, *On the structure of polynomial time reducibility*, J. Assoc. Comput. Mach., 22 (1975), pp. 155–171.

[15] C. H. Papadimitriou and M. Yannakakis, *On limited nondeterminism and the complexity of the V–C dimension*, in 8th Ann. IEEE Conf. Structure in Complexity Theory, 1993, pp. 12–18.

[16] V. R. Pratt, *Every prime has a succinct certificate*, SIAM J. Comput., 4 (1975), pp. 214–220.

[17] L. Sanchis, *Constructing language instances based on partial information*, Internat. J. Foundations Comput. Sci., 5 (1994), pp. 209–229.

SIAM J. COMPUT.
Vol. 27, No. 5, pp. 1430–1437, October 1998
© 1998 Society for Industrial and Applied Mathematics
012

THE PL HIERARCHY COLLAPSES*

MITSUNORI OGIHARA†

Abstract. It is shown that the PL hierarchy $\mathrm{PLH} = \mathrm{PL} \bigcup \mathrm{PL}^{\mathrm{PL}} \bigcup \mathrm{PL}^{\mathrm{PL}^{\mathrm{PL}}} \bigcup \cdots$, defined in terms of the Ruzzo–Simon–Tompa relativization, collapses to PL.

Key words. probabilistic complexity classes, nondeterministic complexity classes, logspace reducibility, constant-depth circuits, relativization

AMS subject classifications. 68Q05, 68Q10, 68Q15

PII. S0097539795295924

1. Introduction. The oracle separations proven by Baker, Gill, and Solovay [2] initiated the study of complexity classes by relativization. Since that work, various relativization models for nondeterministic logspace have been introduced [13, 19, 16, 18]. Ladner and Lynch [13] and I. Simon [19] studied the models that allow oracle machines to make nondeterministic moves while generating query strings and to generate polynomially long query strings. Unfortunately, their models produce some counterintuitive results. With NL as the oracle, NL becomes NP while P remains P. On the other hand, Rackoff and Seiferas [16] studied the model that allows nondeterministic moves during query generation but demands the query strings be logarithmic length-bounded, even for deterministic logspace machines. This model does not produce reverse inclusions, but oracles do not have much power. Even with QBF as the oracle, NL remains NL. To overcome the problems of these models, Ruzzo, J. Simon, and Tompa [18] proposed to allow polynomially long query strings and to demand that nondeterministic Turing machines run deterministically while generating query strings. The Ruzzo–Simon–Tompa model is reasonable. Nonrelativized inclusions of logspace classes such as $\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P}$ hold for every oracle, and powerful oracles make NL bigger, e.g., NL relative to QBF is PSPACE. Because of the reasonability, the Ruzzo–Simon–Tompa model is widely used as the standard relativization model. By extending the restriction to probabilistic computation as well, one can examine classes defined by stacks of logspace classes. But does this make sense? Ruzzo, Simon, and Tompa show that the hierarchy with respect to BPL [10] (bounded-error probabilistic logspace with unlimited computation time) collapses to BPL. Since $\mathrm{NL} \subseteq \mathrm{BPL}$ [10], stacks of NL and BPL never go beyond BPL. Also, the $\mathrm{NL} = \mathrm{coNL}$ theorem [11, 20] has collapsed the NL hierarchy to NL. So, one might wonder whether similar collapse results hold for other logspace classes, in particular, PL. Do we have any intuition concerning stacks of PL? It is known that PL is somewhere between NC^1 and NC^2 [7], but this does not indicate anything about where a stack of PLs will lie. Moreover, stacks of its relative, PP, do not seem to collapse (see, e.g., [5]). But, guessing that $\mathrm{PL}^{\mathrm{PL}} \neq \mathrm{PL}$ by analogy is perhaps too simplistic. In this paper, we resolve the question by showing that $\mathrm{PL}^{\mathrm{PL}} = \mathrm{PL}$, i.e., that the PL hierarchy collapses.

The proof is built on top of the characterization of relativized PL in terms of

* Received by the editors December 8, 1995; accepted for publication (in revised form) August 2, 1996; published electronically May 19, 1998.
http://www.siam.org/journals/sicomp/27-5/29592.html
† Department of Computer Science, University of Rochester, Rochester, NY 14627 (ogihara@cs.rochester.edu). This research was supported in part by the National Science Foundation under grant CCR 9701911.

GapL-functions [1] based on Jung's observation that $\mathrm{PL}_{\mathrm{poly}}$, the polynomial time-bounded version of PL, is identical to PL [12]. GapL is the class of functions that count the difference between the number of accepting computation paths and that of rejecting computation paths of nondeterministic Turing machines (see [8, 1]). Furthermore, the proof makes use of the "polynomial technique" developed in papers [14, 15, 6, 9]. In particular, our proofs borrow much of their construction from that of Fortnow and Reingold [9], who show that PP is closed under polynomial-time truth-table reductions.

This paper is organized as follows. In section 2, we define the class PL and present a useful characterization of the class. Section 3 provides the proof of the collapse result.

2. Preliminaries.

2.1. A characterization of PL. We first define the class PL.

DEFINITION 2.1 (see [10]). *A language L belongs to* PL *if there is a logarithmic space-bounded probabilistic Turing machine M with no time-bound such that for every x, $x \in L$ if and only if the probability that M on x accepts is at least a half.*

An immediate question is whether demanding that the machine M be polynomial time-bounded makes the class smaller. Let $\mathrm{PL}_{\mathrm{poly}}$ denote the class defined similarly to PL, but with a polynomial time-bound on M. Jung [12] showed that these classes are identical. Moreover, the equivalence holds relative to any oracle.

PROPOSITION 2.2 (see [1]). *For every oracle H, $\mathrm{PL}^H = (\mathrm{PL}_{\mathrm{poly}})^H$.*

This equivalence allows us to introduce counting classes that capture PL. For a nondeterministic Turing machine M and a string x, let $acc_M(x)$ (respectively, $rej_M(x)$) denote the number of accepting (respectively, rejecting) computation paths of M on x, and let $gap_M(x)$ denote $acc_M(x) - rej_M(x)$. Define the logspace analogue GapL [1] of GapP [8] as follows.

$$\mathrm{GapL} = \{gap_M \mid M \text{ is a logarithmic space-bounded and polynomial time-bounded nondeterministic Turing machine }\}.$$

By replacing the probabilistic coin tosses by nondeterministic moves, we obtain the following characterization of PL.

PROPOSITION 2.3 (see [1]). *$L \in \mathrm{PL}$ if and only if there exists $f \in \mathrm{GapL}$ such that for every x, $x \in L$ if and only if $f(x) \geq 0$.*

The following properties of GapL can be proven using techniques similar to those used for GapP in [8].

THEOREM 2.4. *Let f be a function in* GapL, *$g : \Sigma^* \times \mathbf{N} \mapsto \Sigma^*$ be a function in* FL, *and p be a polynomial. Then the following functions h_1, h_2, and h_3 all belong to* GapL*:*

1. $h_1(x) = -f(x)$,
2. $h_2(x) = \sum_{i=1}^{p(|x|)} f(g(x,i))$,
3. $h_3(x) = \prod_{i=1}^{p(|x|)} f(g(x,i))$.

Given a function $f \in \mathrm{GapL}$ witnessing that a language L is in PL, define g by $g(x) = 2f(x) + 1$. Then g always takes on odd values and witnesses that L is in PL.

PROPOSITION 2.5. *A language L is in* PL *if and only if there exists a function f in* GapL *such that for every x,*

$$f(x) \geq 1 \text{ if } x \in L \text{ and } f(x) \leq -1 \text{ otherwise.}$$

2.2. The polynomials. Now we define the polynomials that play a crucial role in the proof. These were defined in [6] and [9].

DEFINITION 2.6. *Let $m \geq 1$ and $k \geq 1$. Define polynomials $\mathcal{P}_m(z)$ and $\mathcal{Q}_m(z)$ in $\mathbf{Z}[z]$ by*

$$\mathcal{P}_m(z) = (z-1)\prod_{i=1}^{m}(z-2^i)^2 \quad \textit{and} \tag{1}$$

$$\mathcal{Q}_m(z) = -(\mathcal{P}_m(z) + \mathcal{P}_m(-z)), \tag{2}$$

and define $\mathcal{R}_{m,k}(z)$ and $\mathcal{S}_{m,k}(z)$ by

$$\mathcal{R}_{m,k}(z) = \left(\frac{2\mathcal{P}_m(z)}{\mathcal{Q}_m(z)}\right)^{2k} \quad \textit{and} \tag{3}$$

$$\mathcal{S}_{m,k}(z) = (1+\mathcal{R}_{m,k}(z))^{-1}. \tag{4}$$

Furthermore, define polynomials $\mathcal{A}_{m,k}(z)$ and $\mathcal{B}_{m,k}(z)$ by

$$\mathcal{A}_{m,k}(z) = \mathcal{Q}_m(z)^{2k} \quad \textit{and} \tag{5}$$

$$\mathcal{B}_{m,k}(z) = \mathcal{Q}_m(z)^{2k} + (2\mathcal{P}_m(z))^{2k}. \tag{6}$$

LEMMA 2.7. *For every $m, k \geq 1$ and every z, the following properties hold.*

1. $\mathcal{S}_{m,k}(z) = \mathcal{A}_{m,k}(z)/\mathcal{B}_{m,k}(z)$.
2. *If $1 \leq z \leq 2^m$, then $1 - 2^{-k} \leq \mathcal{S}_{m,k} \leq 1$.*
3. *If $-2^m \leq z \leq -1$, then $0 \leq \mathcal{S}_{m,k}(z) \leq 2^{-k}$.*

Proof. Let $m, k \geq 1$. The first equivalence follows from a simple calculation. Note that $\mathcal{P}_m(z)$ is nonnegative if and only if $z \geq 1$. Let z be ≥ 1. We claim that $\mathcal{P}_m(z) \leq |\mathcal{P}_m(-z)|/4$. This is seen as follows. Clearly, if $z = 1$, then $\mathcal{P}_m(z) = 0 \neq \mathcal{P}_m(-z)$, and thus, the claim holds. On the other hand, if $z > 1$, then there exists a unique $t, 1 \leq t \leq m$, such that $2^t \leq z < 2^{t+1}$, which satisfies $|z - 2^t| \leq z/2 \leq |-z-2^t|/2$. Noting that $|z-1| \leq |-z-1|$ and for every $i, 1 \leq i \leq m$, that $|z-2^i| \leq |-z-2^i|$, we have $\mathcal{P}_m(z) \leq |\mathcal{P}_m(-z)|/4$.

Thus the claim is proven. So, for every z,

$$0 \leq \frac{2\mathcal{P}_m(z)}{\mathcal{Q}_m(z)} \leq \frac{2}{3} \text{ if } 1 \leq z \leq 2^m \text{ and } \frac{2\mathcal{P}_m(z)}{\mathcal{Q}_m(z)} \leq -2 \text{ if } -2^m \leq z \leq -1.$$

Since $(2/3)^2 \leq 1/2$, for every z,

$$0 \leq \mathcal{R}_{m,k}(z) \leq 2^{-k} \text{ if } 1 \leq z \leq 2^m \text{ and } \mathcal{R}_{m,k}(z) \geq 2^k \text{ if } -2^m \leq z \leq -1.$$

Since $\mathcal{S}_{m,k}(z) = (1+\mathcal{R}_{m,k}(z))^{-1}$ and $(1+2^{-k})(1-2^{-k}) < 1$, for every $z, 1 \leq z \leq 2^m$, we have

$$1 - 2^{-k} \leq \mathcal{S}_{m,k}(z) \leq 1.$$

Also, by observing that $(1+2^k)^{-1} \leq 2^{-k}$, we see that $0 \leq \mathcal{S}_{m,k}(z) \leq 2^{-k}$, for all z in the range $[-2^m, -1]$. This proves the lemma. □

3. The PL hierarchy collapses. The following lemma states that logarithmic space-bounded oracle Turing machines can be normalized so that the queries, including the query order, are independent of the oracle.

LEMMA 3.1. *Let $L \in \mathrm{PL}^H$ for some oracle H. Then there exist polynomials p and q and a logarithmic space-bounded nondeterministic Turing machine N such that for every x, the following properties hold.*

1. *On input x, N makes exactly $p(|x|)$ queries and exactly $q(|x|)$ nondeterministic moves, independent of the oracle. Moreover, N on x makes no nondeterministic moves while generating queries, and the queries are independent of the oracle as well.*

2. *For each $i, 1 \leq i \leq p(|x|)$, the ith query of N is computable in logarithmic space from x and i.*

3. *$x \in L$ if and only if $gap_{N^H}(x) > 0$, and $x \notin L$ if and only if $gap_{N^H}(x) < 0$.*

Proof. Let M be the base probabilistic logarithmic space-bounded machine witnessing that $L \in \mathrm{PL}^H$. By Proposition 2.2, we may assume that M is polynomial time-bounded. There is a polynomial q_0 such that for every x, M on x tosses at most $q_0(|x|)$ coins no matter what its oracle may be. By normalizing the number of coin-tosses, without changing the acceptance probability, we can modify M so that M tosses exactly $q_0(|x|)$ coins. On the other hand, since M behaves deterministically while generating queries, each potential query of M on x can be encoded using a string of length $O(\log |x|)$, where the string denotes the ID (the head positions, the state, and the contents of the work-tapes) at which M starts to generate the query. Let r_1 be a polynomial bounding the running time of M and let r_2 be a polynomial bounding the number of queries M could potentially make. Let $p(n) = r_1(n)r_2(n)$. Let M' be a Turing machine that simulates M as follows.

- When M starts to generate a query, say u, M' records the current ID, say I_0.
- When M enters the query state, M' makes all the potential queries of M on x by cycling through all the possible IDs I of M on input x and asking the query generated by each ID. Each time a query is made, M' tests whether $I = I_0$. If $I = I_0$, then M' stores the answer from the oracle in a memory cell. Otherwise, the answer is discarded. Also, M' computes the count C of the number of queries that are made while cycling through all the potential IDs. When all the potential queries have been made, M' compares the count C with $r_2(|x|)$. If $C < r_2(|x|)$, then M' makes additional queries about the empty string $r_2(|x|) - C$ times. The answers to these queries are discarded. With these additional queries, M' will have made exactly $r_2(|x|)$ queries for the ID I_0. Now M' returns to the simulation of M on x with the stored answer.
- The machine M' also keeps the number Q of query rounds that have been made so far. When M enters a halting state, if the number Q is smaller than $r_1(|x|)$, then M' executes the above query round exactly $r_1(|x|) - Q$ times with some fixed ID and discards the answers. With these additional query rounds, M' will have made exactly $r_1(|x|)$ query rounds. Now M' accepts if and only if M has accepted in the simulation.

It is obvious that M' satisfies conditions (1) and (2). Now replace the coin tosses of M' by nondeterministic choices. Let M'' be the resulting machine. The probability that M' on x accepts relative to H is equal to the ratio of the number of accepting computation paths of M'' on x relative to H. Define M_1 to be the Turing machine that, on input x, guesses one bit b and then simulates M'' on x. The number of accepting (rejecting) computation paths of M_1 on x is twice as many as that of M'' on x. Also, define M_2 to be the Turing machine that, on input x, guesses a bit b, simulates M'' on x, and accepts x if and only if either (the bit $b = 0$) or (the bit $b = 1$ and all the nondeterministic guesses for simulating M'' are 0). Then, M_2 on x has exactly $2^{q_0(|x)} + 1$ accepting paths. Now define N to be the machine that, on input x, guesses $i \in \{1, 2\}$, and simulates M_i on x, and accept if and only if M_i on x accepts. Let $q(n) = q_0(n) + 2$. Then N satisfies (1) and (2). It is easy to see that (3)

is satisfied, too. □

THEOREM 3.2. $\mathrm{PL}^{\mathrm{PL}} = \mathrm{PL}$.

Proof. Let $L \in \mathrm{PL}^{\mathrm{PL}}$ be witnessed by a nondeterministic Turing machine N and a language $H \in \mathrm{PL}$ satisfying the conditions in Lemma 3.1 with polynomials p and q. Without loss of generality, we may assume that $p(n), q(n) > 1$ for all n. For each x and $i, 1 \leq i \leq p(|x|)$, let $y_{x,i}$ denote the ith query asked by N on x. Let f be a function in GapL witnessing that $H \in \mathrm{PL}$ as in Proposition 2.5. There exists a polynomial μ such that for every x and $i, 1 \leq i \leq p(|x|)$, $1 \leq |f(y_{x,i})| \leq 2^{\mu(|x|)}$. Let us fix such a polynomial μ. Define $\kappa(n) = p(n) + q(n) + 1$, and define for each x and $i, 1 \leq i \leq p(|x|)$,

$$S(x,i,1) = \mathcal{S}_{\mu,\kappa}(f(y_{x,i})) \text{ and } S(x,i,0) = 1 - \mathcal{S}_{\mu,\kappa}(f(y_{x,i})),$$

where $\mathcal{S}_{\mu,\kappa}$ is short-hand for $\mathcal{S}_{\mu(|x|),\kappa(|x|)}$. By Lemma 2.7, for every x, $i, 1 \leq i \leq p(|x|)$, and $b \in \{0,1\}$,

(7) if $\chi_H(y_{x,i}) = b$, then $1 - 2^{-\kappa(|x|)} \leq S(x,i,b) \leq 1$, and

(8) if $\chi_H(y_{x,i}) \neq b$, then $0 \leq S(x,i,b) \leq 2^{-\kappa(|x|)}$.

Furthermore, define

$$\begin{aligned} \alpha(x,i,1) &= \mathcal{A}_{\mu,\kappa}(f(y_{x,i})), \\ \alpha(x,i,0) &= \mathcal{B}_{\mu,\kappa}(f(y_{x,i})) - \mathcal{A}_{\mu,\kappa}(f(y_{x,i})), \text{ and} \\ \beta(x,i) &= \mathcal{B}_{\mu,\kappa}(f(y_{x,i})). \end{aligned}$$

Then for every x, $i, 1 \leq i \leq p(|x|)$, and $b \in \{0,1\}$,

$$S(x,i,b) = \alpha(x,i,b)/\beta(x,i).$$

For each x and $w \in \{0,1\}^{p(|x|)}$, define

$$\widetilde{S}(x,w) = \prod_{i=1}^{p(|x|)} S(x,i,w_i),$$

where w_i denotes the ith bit of w. Then, by (7) and (8), we have

(9) if $w = \chi_H(y_{x,1}) \cdots \chi_H(y_{x,p(|x|)})$, then $1 - p(|x|)2^{-\kappa(|x|)} \leq \widetilde{S}(x,w) \leq 1$, and

(10) if $w \neq \chi_H(y_{x,1}) \cdots \chi_H(y_{x,p(|x|)})$, then $0 \leq \widetilde{S}(x,w) \leq 2^{-\kappa(|x|)}$.

Define

$$\widetilde{\alpha}(x,w) = \prod_{i=1}^{p(|x|)} \alpha(x,i,w_i) \quad \text{and}$$

$$\widetilde{\beta}(x) = \prod_{i=1}^{p(|x|)} \beta(x,i).$$

Then, for every x and w,

$$\widetilde{S}(x,w) = \widetilde{\alpha}(x,w)/\widetilde{\beta}(x).$$

Define the predicate e as follows:

(11) For each x, $w, |w| = p(|x|)$, and $u, |u| = q(|x|)$, $e(x, w, u) = 1$ if and only if N on input x with nondeterministic guesses u accepts assuming that the answer to the ith query is affirmative if and only if $w_i = 1$.

Define

$$T(x) = \sum_{w,u:|w|=p(|x|),|u|=q(|x|)} e(x,w,u)\widetilde{S}(x,w) \quad \text{and}$$
$$a(x) = \sum_{w,u:|w|=p(|x|),|u|=q(|x|)} e(x,w,u)\widetilde{\alpha}(x,w).$$

Clearly, $T(x) = a(x)/\beta(x)$. By (9) and (10), the following properties hold.

1. For each x, there is a unique $w_x \in \Sigma^{p(|x|)}$ such that $1 - p(|x|)2^{-\kappa(|x|)} \leq \widetilde{S}(x, w_x) \leq 1$ and for every $w \neq w_x$, $0 \leq \widetilde{S}(x, w) \leq 2^{-\kappa(|x|)}$.

2. If $x \in L$, then the number of $u, |u| = q(|x|)$, such that $e(x, w_x, u) = 1$ is at least $2^{q(|x|)-1} + 1$.

3. If $x \notin L$, then the number of $u, |u| = q(|x|)$, such that $e(x, w_x, u) = 1$ is at most $2^{q(|x|)-1} - 1$.

Recall that $\kappa(n) = p(n) + q(n) + 1$ and $p(n), q(n) > 1$. For every x, if $x \in L$, then

$$\begin{aligned} T(x) &\geq 2^{q(|x|)-1}(1 - p(|x|)2^{-\kappa(|x|)}) \\ &\geq (2^{q(|x|)-1} + 1)(1 - 2^{p(|x|)}2^{-\kappa(|x|)}) \\ &= 2^{q(|x|)-1} + (1 - 2^{p(|x|)}2^{-\kappa(|x|)}) - 2^{q(|x|)-1}2^{p(|x|)}2^{-\kappa(|x|)} \\ &\geq 2^{q(|x|)-1} + 1/2 - 1/4 \\ &> 2^{q(|x|)-1}, \end{aligned}$$

so $T(x) > 2^{q(|x|)-1}$, and if $x \notin L$, then

$$\begin{aligned} T(x) &\leq (2^{q(|x|)-1} - 1) + 2^{p(|x|)+q(|x|)}2^{-\kappa(|x|)} \\ &= 2^{q(|x|)-1} - 1 + 2^{-1} \\ &< 2^{q(|x|)-1} - 1/2, \end{aligned}$$

so $T(x) < 2^{q(|x|)-1}$. This implies that for every x, $x \in L$ if and only if $T(x) > 2^{q(|x|)-1}$ and $x \notin L$ if and only if $T(x) < 2^{q(|x|)-1}$. Finally, define $h(x) = 4a(x) - 2^{q(|x|)+1}\beta(x)$. Then, for every x, $x \in L$ if and only if $h(x) \geq 0$.

It remains to show that h is in GapL. Define π to be the function that maps w to $2^{|w|}$. The constant 2 function is trivially in GapL and so, by part (3) of Theorem 2.4, $\pi \in$ GapL. So, one can prove that the function that maps x to $\mathcal{P}_{\mu(|x|)}(f(x))$, i.e., $(f(x) - 1)\prod_{i=1}^{\mu(|x|)}(f(x) - \pi(0^i))^2$, is in GapL by combining the three parts of Theorem 2.4. For much the same reason, the function that maps x to $\mathcal{Q}_{\mu(|x|)}(f(x))$ is in GapL. Since $y_{x,i}$ is logarithmic-space computable, by Theorem 2.4, $\alpha, \beta \in$ GapL. This implies $\widetilde{\beta} \in$ GapL. Since the function that maps x to $2^{q(|x|)+1}$ clearly belongs to GapL, we have only to show that a is in GapL.

Let D be a logarithmic space-bounded nondeterministic Turing machine witnessing that $\alpha \in$ GapL; that is, $\alpha = gap_D$. Note that D on input $\langle x, i, b\rangle$ with $1 \leq i \leq p(|x|)$ and $b \in \{0, 1\}$ can be simulated in $\mathcal{O}(\log |x|)$ space. Define G to be the nondeterministic Turing machine that, on input x, behaves as follows.

Step 1. G first sets a one-bit memory c to 0.

Step 2. G starts simulating N on x nondeterministically; that is, if N makes its ith nondeterministic move, then so does G by guessing bit u_i. When N makes its ith query $y_{x,i}$, G does the following.

(a) G nondeterministically guesses $w_i \in \{0,1\}$ and simulates D on $\langle x, i, w_i\rangle$. If D rejects, then G flips bit c.

(b) G returns to the simulation of N on x assuming that the answer to the query is affirmative if and only if $w_i = 1$.

Step 3. When N enters the halting state, G does the following.

(a) If N has accepted, then G accepts if and only if $c = 0$.

(b) If N has rejected, then G nondeterministically guesses a bit $d \in \{0,1\}$ and accepts if and only if $d = 0$.

Note that, at the beginning of Step 3, $e(x,w,u) = 1$ holds if and only if N has accepted with w and u. In the case when N has rejected, and thus, $e(x,w,u) = 0$, G generates one accepting path and one rejecting path, so there is no contribution to $gap_G(x)$ from this w and u. In the case when N has accepted, and thus, $e(x,w,u) = 1$, the one-bit memory c is the parity of the number of accepting simulations of D that G has encountered. Since G accepts if and only if the parity is 0, the number of accepting computation paths along w and u is the sum of all

$$\prod_{i \notin I} acc_D(x,i,w_i) \prod_{i \in I} rej_D(x,i,w_i)$$

with I ranging over all subsets of $\{1, \ldots, p(|x|)\}$ of even cardinality. Also, the number of rejecting computation paths generated along w and u is the sum of all

$$\prod_{i \notin I} acc_D(x,i,w_i) \prod_{i \in I} rej_D(x,i,w_i)$$

with I ranging over all subsets of $\{1, \ldots, p(|x|)\}$ of odd cardinality. Note, for every i and w_i, that $acc_D(x,i,w_i) - rej_D(x,i,w_i) = gap_D(x,i,w_i)$. Thus, the difference between the above two sums is equal to

$$\prod_{i=1}^{p(|x|)} (acc_D(x,i,w_i) - rej_D(x,i,w_i)) = \prod_{i=1}^{p(|x|)} gap_D(x,i,w_i).$$

Thus, for every x,

$$\begin{aligned} gap_G(x) &= \sum_{w,u:|w|=p(|x|)|u|=q(|x|)} e(x,w,u) \prod_{i=1}^{p(|x|)} \alpha(x,i,w_i) \\ &= \sum_{w,u:|w|=p(|x|),|u|=q(|x|)} e(x,w,u)\widetilde{\alpha}(x,w) \\ &= a(x). \end{aligned}$$

Hence, a is in GapL. This proves the theorem. □

Allender and Ogihara [1] observe that the PL hierarchy coincides with the logspace uniform AC^0 closure of PL. So, we immediately obtain the following corollary.

COROLLARY 3.3. $PLH = AC^0(PL) = PL$.

The closure can be strengthened further. Given any constant-depth circuit, by slicing the circuits into levels and by using the fact that $\mathrm{PL}^{\mathrm{PL}} = \mathrm{PL}$, one can collapse the two levels into one.

COROLLARY 3.4. PL *is closed under logspace-uniform, constant-depth reductions with arbitrary symmetric functions as gates.*

In particular, PL is closed under logspace-uniform TC^0-reductions [17] and under logspace-uniform ACC-reductions [3].

Can we extend the closure property to nonconstant-depth circuits? Recently, Beigel and Fu have reported that the answer is positive: PL is closed under NC^1-reductions [4]. The result leaves us with the question of whether logarithmic-depth semiunbounded fan-in circuits SAC^1 are in PL, and, if so, whether PL is closed under SAC^1-reductions.

Acknowledgment. The author thanks Eric Allender, Richard Beigel, Lance Fortnow, Janos Simon, and Marius Zimand for enjoyable discussions, and anonymous referees for invaluable comments.

REFERENCES

[1] E. ALLENDER AND M. OGIHARA, *Relationships among PL, #L, and the determinant*, in Proc. 9th IEEE Conf. on Structure in Complex. Theory, 1994, pp. 267–278.
[2] T. BAKER, J. GILL, AND R. SOLOVAY, *Relativizations of the $\mathcal{P}$ =?$\mathcal{NP}$ question*, SIAM J. Comput., 4 (1975), pp. 431–442.
[3] D. BARRINGTON, *Bounded-width polynomial-size branching programs recognize exactly those languages in* NC^1, J. Comput. System Sci., 38 (1989), pp. 150–164.
[4] R. BEIGEL AND B. FU, *Circuits over PP and PL*, in Proc. 12th Conf. on Comput. Complexity, IEEE Computer Society Press, Los Alamitos, CA, 1997, pp. 24–35.
[5] R. BEIGEL, *Perceptrons, PP, and the polynomial hierarchy*, Comput. Complexity, 4 (1994), pp. 339–349.
[6] R. BEIGEL, N. REINGOLD, AND D. SPIELMAN, *PP is closed under intersection*, J. Comput. System Sci., 50 (1995), pp. 191–202.
[7] A. BORODIN, S. COOK, AND N. PIPPENGER, *Parallel computation for well-endowed rings and space-bounded probabilistic machines*, Inform. and Control, 58 (1983), pp. 113–136.
[8] S. FENNER, L. FORTNOW, AND S. KURTZ, *Gap-definable counting classes*, J. Comput. System Sci., 48 (1994), pp. 116–148.
[9] L. FORTNOW AND N. REINGOLD, *PP is closed under truth-table reductions*, Inform. and Comput., 124 (1996), pp. 1–6.
[10] J. GILL, *Computational complexity of probabilistic Turing machines*, SIAM J. Comput., 6 (1977), pp. 675–695.
[11] N. IMMERMAN, *Nondeterministic space is closed under complementation*, SIAM J. Comput., 17 (1988), pp. 935–938.
[12] H. JUNG, *On probabilistic time and space*, in Proc. 12th Conf. on Automata, Lang. and Prog., Lecture Notes in Comp. Sci. 194, Springer-Verlag, New York, 1985, pp. 310–317.
[13] R. LADNER AND N. LYNCH, *Relativization of questions about logspace computability*, Math. Systems Theory, 10 (1976), pp. 19–32.
[14] D. NEWMAN, *Rational approximation to $|x|$*, Michigan Math. J., 11 (1964), pp. 11–14.
[15] S. PATURI AND M. SAKS, *Approximating threshold circuits by rational functions*, Inform. and Comput., 112 (1994), pp. 257–272.
[16] C. RACKOFF AND J. SEIFERAS, *Limitations on separating nondeterministic complexity classes*, SIAM J. Comput., 10 (1981), pp. 742–745.
[17] J. REIF AND S. TATE, *On threshold circuits and polynomial computation*, SIAM J. Comput., 21 (1992), pp. 896–908.
[18] W. RUZZO, J. SIMON, AND M. TOMPA, *Space-bounded hierarchies and probabilistic computations*, J. Comput. System Sci., 28 (1984), pp. 216–230.
[19] I. SIMON, *On Some Subrecursive Reducibilities*, Ph.D. Thesis, Stanford University, Stanford, CA, 1977. Available as Computer Science Department, Stanford University, Technical Report STAN-CS-77-608.
[20] R. SZELEPCSÉNYI, *The method of forced enumeration for nondeterministic automata*, Acta Inform., 26 (1988), pp. 279–284.

SIAM J. COMPUT.
Vol. 27, No. 5, pp. 1438–1456, October 1998
© 1998 Society for Industrial and Applied Mathematics
013

GENERATING LOW-DEGREE 2-SPANNERS*

GUY KORTSARZ† AND DAVID PELEG‡

Abstract. A k-spanner of a connected (undirected unweighted) graph $G = (V, E)$ is a subgraph G' consisting of all the vertices of V and a subset of the edges, with the additional property that the distance between any two vertices in G' is larger than that distance in G by no more than a factor of k. This paper is concerned with approximating the problem of finding a 2-spanner in a given graph, with minimum maximum degree. We first show that the problem is at least as hard to approximate as set cover. Then a randomized approximation algorithm is provided for this problem, with approximation ratio of $\tilde{O}(\Delta^{1/4})$. We then present a probabilistic algorithm that is more efficient for sparse graphs. Our algorithms are converted into deterministic ones using derandomization.

Key words. graph spanners, NP-hardness, approximation, randomized rounding

AMS subject classifications. 05C05, 05C12, 05C85, 68Q25, 68R10, 90C35

PII. S0097539794268753

1. Introduction.

The concept of *graph spanners* has been studied in several recent papers in the context of communication networks, distributed computing, robotics, and computational geometry [ADDJ90, Cai91, Che86, DFS87, DJ89, LL89, PS89, PU89, LR94, CDNS92]. Consider a connected simple (unweighted) graph $G = (V, E)$, with $|V| = n$ vertices. A subgraph $G' = (V, E')$ of G is a *k-spanner* if for every $u, v \in V$,

$$\frac{\text{dist}(u, v, G')}{\text{dist}(u, v, G)} \leq k,$$

where $\text{dist}(u, v, G')$ denotes the distance from u to v in G', i.e., the minimum number of edges in a path connecting them in G'. We refer to k as the *stretch factor* of G'.

In the Euclidean setting, spanners were studied in [Cai91, DFS87, DJ89, LL89, Soa92]. Spanners for general graphs were first introduced in [PU89], where it was shown that for every n-vertex hypercube there exists a 3-spanner with no more than $7n$ edges and then studied further in [PS89, LR94, ADDJ90, CDNS92]. Spanners were used in [PU89] to construct a new type of synchronizer for an asynchronous network.

The usual criteria for the quality of the spanner are its *stretch* and its *sparsity*. Namely, a good spanner is one with low stretch and as few edges as possible. For the problem of finding a 2-spanner which is as sparse as possible, a logarithmic-ratio approximation is given in [KP94].

However, another parameter of significance when selecting a good spanner is the maximum degree of the spanner. In terms of applications, a high degree might mean a high *local load* on a single vertex, increasing the cost of its local management. For instance, in the application of using a spanner for implementing a δ-synchronizer in a distributed network [PU89], or when using a spanner for efficient broadcast [ABP91],

*Received by the editors June 3, 1994; accepted for publication (in revised form) August 9, 1996; published electronically May 19, 1998.

http://www.siam.org/journals/sicomp/27-5/26875.html

†Department of Computer Science, The Open University, Ramat-Aviv, Israel (guyk@tavor.openu.ac.il). Part of this work was done while the author was at the Weizmann Institute.

‡Department of Applied Mathematics and Computer Science, The Weizmann Institute, Rehovot 76100, Israel (peleg@widsom.weizmann.ac.il). This research was supported in part by a Walter and Elise Haas Career Development Award and by a grant from the Israel Science Foundation.

the degree of each node in the spanner directly translates into memory requirements at the node, and high local degrees mean higher workload on the involved nodes.

It is clear that focusing on optimizing the sparsity measure alone may result in a spanner with high vertex degrees. For example, if there is a vertex v in the graph that is connected to all the rest of the vertices, then its edges form a 2-spanner and, therefore, a k-spanner for any $k \geq 2$. However, in such a choice the local load of v might be too high to handle, while the local load of any other vertex w may be much less than what w can handle. In fact, the algorithm proposed in [KP94] will pick the vertex v and is therefore unsuitable for selecting "balanced load" spanners. It is therefore natural to try to design an algorithm that will perform a more "balanced" selection of the edges. In particular, letting $\Delta(G')$ denote the maximum vertex degree in a spanner G', we consider the question of choosing a k-spanner G' with minimum $\Delta(G')$ for some parameter k. We call this "low degree" variant of the problem LD-kSP. The problem of designing low degree spanners is addressed in [LR94, LS93b, LS93a] for some special graph classes such as pyramids and grids. The problem of designing small degree spanners for Euclidean and geometric graphs is studied in [CDNS92, Soa92]. The distance is measured therein by the appropriate *norm* defined in the vector space.

This paper treats LD-2SP in general graphs. We first show that the problem is at least as hard to approximate as set cover (up to constants). This implies the following results. There is no $\ln n/5$-ratio approximation algorithm for LD-2SP unless $NP \subset DTIME(n^{polylog(n)})$. Also, no approximation algorithm with constant ratio exists for the problem unless $P = NP$. We next give a probabilistic algorithm that outputs a 2-spanner G' such that with high probability $\Delta(G')$ is no more than $\tilde{O}(\Delta^{1/4})$ times the optimum. In other words, our algorithm has an approximation ratio of $\tilde{O}(\Delta^{1/4})$ with high probability. ($\tilde{O}$ is a relaxed variant of the usual O notation that ignores polylogarithmic factors.) The algorithm is then turned into a deterministic one using derandomization.

The technique used in [KP94] to approximate the sparsest 2-spanner problem is the "greedy" method that constructs the spanner gradually, attempting to 2-span a large number of edges in every iteration. (An edge $e = (u, v)$ is 2-*spanned* once either itself or two other edges lying on a triangle with it, say (u, x) and (x, v), are added to the spanner.) The LD-2SP problem seems to be harder to approximate. In particular, the greedy approach seems to fail (i.e., be inefficient) for it. Hence a different (and more involved) approach is required. The technique used in this paper for the LD-2SP problem is a variant of the "randomized rounding" technique of [RT87].

Our algorithm is composed of two different procedures. The first procedure is designed to 2-span edges lying on "many" triangles. The second procedure deals with the yet unspanned edges, i.e., edges that lie on a "small" number of triangles. We describe the "2-spanning" problem for these edges as a linear program, solve it in the fractional setting, and randomly round the fractional solutions. We note that in the rounding process, we use only a subset of the variables. We also note that every variable is rounded with probability considerably exceeding its fractional value. These higher rounding probabilities seem to be needed in order to overcome some "quadratic" behavior of the linear program.

We also present an additional probabilistic algorithm that is efficient for *sparse* graphs. This algorithm can also be transformed into a deterministic one using derandomization.

Finally we deal with the problem of 2-spanning only the edges adjacent to a

small subset $V_k, |V_k| = k$ of the vertices. We give an $O(k \cdot \log n)$-ratio approximation algorithm for this problem. For fixed k, our hardness result implies that unless $NP \subset DTIME(n^{\log \log n})$ this is the best ratio possible (asymptotically).

2. Preliminaries. We start by introducing some definitions. In the sequel, let $G = (V, E)$ be the underlying n-vertex graph. We sometimes use E also to denote the *size* of the set E, i.e., the number of edges. Let $U \subseteq V$ be a subset of the vertices. The graph induced by U is denoted by $G(U)$. The set of edges in $G(U)$ is denoted by $E(U)$. For a vertex v we denote by $E(v)$ the set of edges adjacent to v in G. Similarly, we denote by $N(v)$ the set of neighbors of v in G, i.e.,

$$N(v) = \{u \mid (u, v) \in E\}.$$

We denote the degree of a vertex v by $\deg(v) = |N(v)|$. The maximum degree in a subgraph $G' = (V', E')$, where $V' \subset V$ and $E' \subseteq E$, is denoted by $\Delta(G')$. We denote by $\Delta(E')$ the largest degree in the subgraph (V, E'). We sometimes write Δ for $\Delta(G)$.

We make use of an alternative characterization of k-spanners given in the following simple lemma of [PS89].

LEMMA 2.1 (see [PS89]). *The subgraph $G' = (V, E')$ is a k-spanner of the graph $G = (V, E)$ iff $\text{dist}(u, v, G') \leq k$ for every $(v, u) \in E$.* □

Thus the LD-2SP problem can be restated as follows: we look for a subset of edges $E' \subset E$ such that every edge e that does not belong to E' lies on a triangle with two edges that do belong to E' and such that $\Delta(E')$ is minimum.

Given an edge $e \in E$, let $Tri(e)$ denote the set of triangles e lies on in the graph G. Namely,

$$Tri(e) = \{\{e, e_1, e_2\} \mid e, e_1 \text{ and } e_2 \text{ form a triangle in } G\}.$$

Let $D(e)$ be the set of vertices that lie on a triangle with e but do not touch e. (Note that $|D(e)| = |Tri(e)|$, as each vertex in $D(e)$ corresponds to exactly one triangle in $Tri(e)$.) We say that a vertex $v \in D(e)$ (sharing a triangle T with e) 2-*helps* e in the spanner H if the two edges incident to v on T are chosen into H.

In the sequel we estimate the probability of the deviation of some random variables from their expectation, using the Chernoff bound [Che52].

LEMMA 2.2 (see [Che52]). *Let $X_1, X_2, \ldots, X_m$ be independent Bernoulli trials with $\mathbb{P}(X_i = 1) = p_i$. Let $X = \sum_{i=1}^{m} X_i$ and $\mu = \sum_{i=1}^{m} p_i$. Then*

$$\mathbb{P}(X > (1 + \delta)\mu) < \left[\frac{e^{\delta}}{(1 + \delta)^{(1+\delta)}}\right]^{\mu}.$$ □

In the sequel we assume that $\Delta(G) \geq \Omega(\log^2 n)$. If this is not the case, then taking the entire graph as our spanner results in a polylogarithmic-ratio approximation.

Unless stated otherwise, all logarithms in this paper are taken to the base 2.

3. Basic properties.

3.1. Low degree spanners for general graphs and special graph families. The problem of designing low degree spanners is addressed in [LR94] for the special case where the underlying graph is the *pyramid*. In particular, it is proven therein that this graph enjoys a 2-spanner (respectively, 3, 7) with maximum degree 6 (respectively, 4, 3). The problem of designing small degree spanners for Euclidean and geometric

graphs is studied in [CDNS92, Soa92]. There, however, the distance is measured by the appropriate *norm* defined in the vector space.

We now establish some basic properties concerning the degrees of 2-spanners. The next lemma indicates that for a graph with large Δ, the minimum degree in a 2-spanner must also be large. We prove this by showing that even for the sake of 2-spanning the edges of a *single* vertex v with degree Δ, it is necessary to have a vertex in the spanner with degree at least $\sqrt{\Delta}$.

LEMMA 3.1. *Let v be a vertex of degree d in G. Let $H = (V, E(H))$ be a 2-spanner of G. Then in H either v or some vertex in $N(v)$ has degree at least $\sqrt{d}$.*

Proof. Let t denote the maximum degree of any vertex from $N(v)$ in H. Then the number of vertices reachable from v in two steps over H edges is at most t^2. Since all d edges incident to v must be 2-spanned in H, necessarily $t^2 \geq d$. ∎

As an immediate conclusion we have the following.

LEMMA 3.2. *Let $H^* = (V, E^*)$ be a 2-spanner for G with minimum maximum degree. Then*

$$\Delta(H^*) \geq \sqrt{\Delta(G)}. \qquad \blacksquare$$

Let us remark that a similar result holds for k-spanners H^* of minimum maximum degree for any $k \geq 2$, namely, $\Delta(H^*)$ is $\Omega(\Delta(G)^{1/k})$ (the proof is also similar).

Note that there are graphs G for which $\Delta(H) = \Delta(G)$ for any 2-spanner H of G. One particular such graph is the star of $n-1$ vertices. However, there are dense graphs, where the lower bound $\sqrt{\Delta}$ can be achieved (up to constants). The clique (complete graph) K_n of n vertices admits low degree 2-spanners. In order to prove this, we use the notion of a *projective plane of order q* for prime q. The existence of projective planes of order q for every prime q is well known. A projective plane $\mathcal{P} = (P, L)$ of order q is composed of a collection $P = \{p_1, \ldots, p_m\}$ of points and a collection $L = \{l_1, \ldots, l_m\}$ of lines where $m = q^2 + q + 1$. Every line l_i is a subset of P containing exactly $d = q + 1$ points and every point is contained in exactly d lines. Every two lines intersect in exactly one point and every two points share exactly one line.

Consider now $K_n = (V, V \times V)$ where $V = \{v_1, \ldots, v_n\}$. Let q be a prime number such that $\lfloor\sqrt{n}\rfloor \leq q \leq 2\lfloor\sqrt{n}\rfloor$. (Such a prime exists by Bertrand's postulate; cf. [HW56].) Thus, $n < q^2 + q + 1 < 5n$. Let $\mathcal{P} = (P, L)$ be a projective plane of order q. Define the following spanning subgraph $H = (V, E')$ of K_n. Add the edge (v_i, v_j) to E' iff there exist t and r that satisfy $t \equiv i \bmod n$ and $r \equiv j \bmod n$ such that either $p_t \in l_r$ or $p_r \in l_t$.

We now proceed to prove that H is a low degree 2-spanner for K_n. First we note the following claim.

CLAIM 3.3. *The subgraph H is a 2-spanner for K_n.*

Proof. Let $e = (v_i, v_j)$ be an arbitrary edge of K_n. The lines l_i and l_j share some point $p_s \in l_i \cap l_j$. Let f be the integer satisfying $1 \leq f \leq n$, $f \equiv s \bmod n$. If $f = i$ or $f = j$ (i.e., the case is, for example, that $p_j \in l_i \cap l_j$) then by definition $(v_i, v_j) \in E'$. Otherwise, again by definition, both $(v_i, v_f) \in E'$ and $(v_j, v_f) \in E'$, and the edge e is spanned. ∎

We now estimate the degree of the vertices in H'. Note that the degree of a vertex $v_i \in H$ is only increased due to vertices in the set $S_i = \{l_j \mid j \equiv i \bmod n\} \cup \{p_j \mid j \equiv i \bmod n\}$, and $|S_i| \leq 10$. Each vertex in S_i increases the degree of v_i by at most $d = q + 1$, and thus the degree of v_i in H is bounded by $O(d) = O(\sqrt{n})$.

In conclusion, we have established the following claim.

LEMMA 3.4. *The complete graph K_n admits a 2-spanner H with $\Delta(H) = O(\sqrt{\Delta(K_n)})$.* □

4. A hardness result for approximating LD-2SP. In this section we establish that the LD-2SP problem is (up to a constant factor) at least as hard to approximate as the *set cover* problem. Formally, the set cover problem is defined as follows. Given a bipartite graph $G(V_1, V_2, E)$ with $|V_1| = |V_2| = n$, find a minimum cardinality subset S of V_1, that *covers* V_2, i.e., such that every vertex in V_2 has a neighbor in S.

It is known that this problem is hard to approximate. In particular, the following theorem is proved in [LY93, Fei96].

THEOREM 4.1 (see [Fei96]). *The set cover problem cannot be approximated with ratio $\ln n - \epsilon$, for any fixed $\epsilon > 0$, unless $NP \subset DTIME(n^{\log\log n})$.* □

Also, the following theorem is proven in [BGLR93].

THEOREM 4.2 (see [BGLR93]). *The set cover problem cannot be approximated with any constant ratio c, unless $P = NP$.* □

For our purpose, we need a slightly different version of the set cover problem. Define the *$\sqrt{n}$-set cover* problem as a variant of the set cover problem in which $d(v) \leq \sqrt{n}$ for each vertex $v \in V_1 \cup V_2$. The usual greedy algorithm approximates this problem with ratio $\ln\sqrt{n} + 1 = \ln n/2 + 1$ [Joh74, Lov75]. On the other hand, a simple observation gives the following fact.

FACT 4.3. *The $\sqrt{n}$-set cover problem cannot be approximated with ratio better than $\ln n/2$, unless $NP \subset DTIME(n^{\log\log n})$.*

Proof. Assume the existence of an approximation algorithm A for the $\sqrt{n}$-set cover problem, with ratio $\ln n/2$ or better. Let $G(V_1, V_2, E)$ be an instance of the set cover problem. Let $\tilde{G}$ be a graph consisting of n (separate) copies of G. The graph $\tilde{G}$ contains n^2 vertices on each side, and the maximum degree in $\tilde{G}$ is bounded by n. Thus, $\tilde{G}$ is amenable to approximation by algorithm A, and, consequently, the set cover instance represented by $\tilde{G}$ can be approximated with ratio better than $\ln(n^2)/2 = \ln n$. Since any cover in $\tilde{G}$ is composed of n separate covers of V_2, we get by a straightforward averaging argument that one of these covers approximates the optimum cover of V_2 by a ratio better than $\ln n$. By Theorem 4.1, this implies $NP \subset DTIME(n^{\log\log n})$. □

In the remainder of this section, we consider the $\sqrt{n}$-set cover problem, with $|V_1| = |V_2| = n$. We show that the LD-2SP problem is at least as hard to approximate as this problem. Throughout, we denote by t^* the size of the optimum cover of V_2 in G. Let $\ell = \lceil\sqrt{n}\,\rceil$. Note that

$$t^* \geq \ell. \tag{1}$$

4.1. The construction. We use an auxiliary graph $\bar{G}$ constructed from G as follows. The vertices of $\bar{G}$ are

$$V_1 \cup V_2 \cup \{s\} \cup \{c(v_1) \mid v_1 \in V_1\} \cup \{u_1, \ldots, u_\ell\}.$$

Divide the vertices of V_1 arbitrarily into ℓ disjoint sets V_1^i, where each V_1^i contains no more than $\sqrt{n}$ vertices. The edge set of $\bar{G}$ is given by defining a number of edge

classes as follows. Let

$$\begin{aligned}
\mathcal{E}_1 &= E(G),\\
\mathcal{E}_2 &= \{(s, v_1) \mid v_1 \in V_1\},\\
\mathcal{E}_3 &= \{(s, v_2) \mid v_2 \in V_2\},\\
\mathcal{E}_4 &= \{(c(v_1), v_1) \mid v_1 \in V_1\} \cup \{(c(v_1), v_2) \mid (v_1, v_2) \in E(G)\},\\
\mathcal{E}_5 &= \{(u_i, v_1) \mid v_1 \in V_1^i, 1 \le i \le \ell\},\\
\mathcal{E}_6 &= \{(s, u_i) \mid 1 \le i \le \ell\},
\end{aligned}$$

and set

$$E(\bar{G}) = \mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{E}_3 \cup \mathcal{E}_4 \cup \mathcal{E}_5 \cup \mathcal{E}_6.$$

Let us make a few remarks on this construction. As the maximum degree in $\bar{G}$ is more than $2n$, the best 2-spanner H of G has maximum degree at least $\sqrt{2n}$. We note that for the sake of choosing a good spanner, one can "afford" the edges of classes $\mathcal{E}_1$, $\mathcal{E}_4$, $\mathcal{E}_5$, and $\mathcal{E}_6$, since it easily follows from the construction that these edges induce a subgraph with maximum degree bounded by ℓ. This observation relies on the fact that the degrees of the vertices of V_1 and V_2 in G are bounded by $\sqrt{n}$.

Consequently, the edges from which a "good" spanner may need to omit are the edges of classes $\mathcal{E}_2$ and $\mathcal{E}_3$. These edges give s its degree in $\bar{G}$, which is higher than $2n$. The heart of our proof lies in 2-spanning the edges of $\mathcal{E}_3$. Such an edge e must either be included in the spanner or be 2-helped by some vertex $v_1 \in D(e)$. In order to keep the degree of s low, one has to choose a small subset of V_1 that covers V_2.

4.2. The main claim. We prove the main result of this section using the following two lemmas.

LEMMA 4.4. *If G contains a cover of size t^* then $\bar{G}$ contains a 2-spanner with maximum degree bounded by $2t^*$.*

Proof. Assume a graph $G(V_1, V_2, E)$ with a t^*-cover $C = \{v_1, \ldots, v_{t^*}\} \subseteq V_1$ of V_2. For every vertex $v_2 \in V_2$ choose a vertex $R(v_2) \in C$ connected to v_2. Create a spanner H consisting of the following two edge sets:

$$\begin{aligned}
\mathcal{H}_1 &= \{(v_2, R(v_2)) \mid v_2 \in V_2\} \cup \{(s, v_1) \mid v_1 \in C\},\\
\mathcal{H}_2 &= \mathcal{E}_4 \cup \mathcal{E}_5 \cup \mathcal{E}_6,
\end{aligned}$$

and set $E(H) = \mathcal{H}_1 \cup \mathcal{H}_2$.

(*Remark.* The above construction can be slightly improved. For example, it is not necessary to put in H edges $(c(v_1), u)$ if v_1 is the vertex chosen to cover u. We note, nevertheless, that the maximum degree is not decreased by these improvements.)

We need the following observations.

CLAIM 4.5. *The maximum degree in H is bounded by $2t^*$.*

Proof. As explained before, the edges of classes $\mathcal{E}_4$, $\mathcal{E}_5$, and $\mathcal{E}_6$ induce a graph with maximum degree bounded by ℓ. Specifically, s, the $c(v_i)$ vertices and the vertices of V_2 have degree bounded by ℓ and the vertices of V_1 have degree bounded by 2 (i.e., a vertex v_1 is connected to its u_i and to $c(v_1)$.)

Now consider the edges of $\mathcal{H}_1$. The degree of each vertex in C is increased by no more than $\sqrt{n}+1$ due to these edges (these edges connect a vertex v_1 to its neighbors in V_2 and to s). Note, however, that t^* new edges are added to s (connecting s to the vertices of C), thus making its degree no more than $t^* + \ell \le 2t^*$ by inequality (1). Hence the maximum degree is as stated. ∎

Next we establish the following claim.

CLAIM 4.6. *The graph H is a 2-spanner of $\bar{G}$.*

Proof. The edges of classes $\mathcal{E}_4$, $\mathcal{E}_5$, and $\mathcal{E}_6$ are in the spanner. We must show that the edges of classes $\mathcal{E}_1$, $\mathcal{E}_2$, and $\mathcal{E}_3$ are 2-spanned by H.

An edge $(v_1, v_2) \in \mathcal{E}_1$ is 2-spanned in the spanner by the edges $(c(v_1), v_1)$ and $(c(v_1), v_2)$, since both are in $\mathcal{E}_4$.

Next consider an edge $(v_1, s) \in \mathcal{E}_2$. The node v_1 must belong to some set V_1^i. Therefore this edge is 2-spanned in H by the edges (u_i, v_1) and (u_i, s), which are in $\mathcal{E}_5$ and $\mathcal{E}_6$, respectively.

Finally, an edge $(v_2, s) \in \mathcal{E}_3$ is 2-spanned in H by the edges $(v_2, R(v_2))$ and $(R(v_2), s)$, which are in $\mathcal{H}_1$. □

The second central lemma in this section is the following.

LEMMA 4.7. *Given a 2-spanner H of $\bar{G}$, it is possible to find a cover of V_2 in G with cardinality bounded by $\Delta(H)$.*

Proof. Consider a 2-spanner H. Partition the edges of class $\mathcal{E}_3$ into two disjoint sets

$$\mathcal{E}_3^A = \mathcal{E}_3 \cap E(H) \quad \text{and} \quad \mathcal{E}_3^B = \mathcal{E}_3 \setminus E(H).$$

Note that each of the edges of $\mathcal{E}_3^B$ is 2-spanned in H by a 2-helping vertex.

Construct a cover of V_2 in G as follows. First, for every edge $(v_2, s) \in \mathcal{E}_3^A$, choose an arbitrary vertex v_1 connected to v_2 in G, and let D_1 be the set of selected vertices. Second, for every edge (s, v_2) in $\mathcal{E}_3^B$, choose an arbitrary vertex $v_1 \in V_1$ such that both (s, v_1) and (v_2, v_1) are in $E(H)$. (A single vertex of V_1 may be chosen several times, as it may cover many edges in $\mathcal{E}_3^B$.) Let D_2 be the set of vertices chosen by this process.

Clearly, the set $D_1 \cup D_2$ forms a cover of V_2 in G. It remains to bound its cardinality. First note that $|D_1| \leq |\mathcal{E}_3^A|$ (at worst, a *different* vertex v_1 is chosen to D_1 for every vertex of $\mathcal{E}_3^A$). Now, every edge in $\mathcal{E}_3^A$ adds 1 to the degree of s in H. Also, every vertex in D_2 is connected to s in H and thus adds 1 to its degree. Hence $|D_1| + |D_2| \leq \Delta(H)$. This proves the claim. □

The following corollary is now immediate.

COROLLARY 4.8. *The* LD-2SP *problem cannot be approximated with ratio better than $\ln n/5$, unless $NP \subset DTIME(n^{\log \log n})$.*

Proof. Assume the existence of an approximation algorithm A for the LD-2SP problem, with ratio better than $\ln n/5$. Take an input G of the $\sqrt{n}$-set cover problem. Let t^* be the size of a minimum cover of V_2 in G. Construct $\bar{G}$ as explained before (this can clearly be done in polynomial time). By Lemma 4.4 the graph $\bar{G}$ contains a 2-spanner with maximum degree bounded by $2t^*$. Note that the number of vertices in $\bar{G}$ is less than $4n$. By the assumption, it is possible to use algorithm A and find a 2-spanner with maximum degree bounded by

$$2t^*(\ln |\bar{G}|)/5 \leq 2t^*(\ln n + 4)/5 < t^*(\ln n/2 - 1)$$

(the last inequality holds for sufficiently large n). By Lemma 4.7 this implies that it is possible to find in polynomial time a cover of V_2 in G with cardinality bounded by $t^*(\ln n/2 - 1)$. By Fact 4.3, this implies $NP \subset DTIME(n^{\log \log n})$. □

The following corollary also follows easily.

COROLLARY 4.9. *The* LD-2SP *problem cannot be approximated with ratio c, for any constant c, unless $P = NP$.* □

5. The approximation algorithm for LD-2SP. Let us first explain the idea behind our approximation algorithm for the LD-2SP problem. We separate the edge set of our graph into two disjoint classes. The class E^- is the class of edges that lies on a small number of triangles, and the class E^+ contains the rest of the edges, namely,

$$E^- = \{e \in E \mid Tri(e) < \sqrt{\Delta}\}, \qquad E^+ = \{e \in E \mid Tri(e) \geq \sqrt{\Delta}\}.$$

(Recall that $Tri(e)$ is the set $\{e, e_1, e_2\}$ of triangles containing e.) We 2-span these two classes of edges, using two separate procedures. Our general approach for handling E^- is to use the "randomized rounding" scheme of Raghavan and Thompson [RT87]. This scheme is based on the following idea. Let Δ^* be the maximum degree in the best 2-spanner. (We shall soon see that it is possible, without loss of generality, to assume that Δ^* is known.) We then formulate the problem as the integer linear program $(P1)$ below.

The program ($P1$). Given are some subset $E_u \subseteq E$ of "unspanned" edges and a (possibly empty) set E_r of edges that have already been added to the spanner. Create for every edge $e_k \in E_u$ and vertex $v_i \in D(e_k)$ a variable $\hat{y}_{i,k}$. (Recall that $D(e)$ is the set of vertices that lies on a triangle with e but does not touch it.) For every two vertices $v_i, v_j \in V, i < j$, such that $(v_i, v_j) \in E$, create a variable $\hat{x}_{i,j}$. (We shall freely use both $\hat{x}_{i,j}$ and $\hat{x}_{j,i}$ to denote this unique variable.) The program is composed of the following sets of inequalities:

$$\sum_{v_j \in N(v_i), (v_i,v_j) \notin E_r} \hat{x}_{i,j} \leq \Delta^* \quad \text{for all } v_i \in V, \tag{2}$$

$$\hat{x}_{l,t} + \sum_{v_i \in D(e_k)} \hat{y}_{i,k} \geq 1 \quad \text{for all } e_k = (v_l, v_t) \in E_u, \tag{3}$$

$$\hat{y}_{i,k} \leq \hat{x}_{i,l}, \hat{x}_{i,t} \quad \text{for all } e_k = (v_l, v_t) \in E_u \text{ and } v_i \in D(e_k), \tag{4}$$

$$\hat{x}_{ij} = 1 \quad \text{for all } e = (v_i, v_j) \in E_r, \tag{5}$$

$$\hat{x}_{i,j},\ \hat{y}_{i,k} \in \{0,1\} \quad \text{for all } i, j, k. \tag{6}$$

The intuitive meaning of the program is as follows. Every $\hat{x}_{i,j}$ variable indicates if the edge (v_i, v_j) is in the chosen spanner. Thus constraint (2) says that every vertex v_i has no more than Δ^* new spanner edges. It is important to note that here we do not count the edges of E_r (those edges are counted separately in the analysis). The variable $\hat{y}_{i,k}$, associated with a vertex v_i and an edge $e_k = (v_l, v_t)$, indicates if v_i 2-helps e_k in the chosen spanner. This is enforced by constraint (4), which says that v_i 2-helps e_k only if both the edges (v_i, v_l) and (v_i, v_t) are included in the spanner. Constraint (3) says that in a feasible 2-spanner, every edge is either in the spanner or is 2-helped by some vertex.

After writing the program, we solve the fractional relaxation of $(P1)$ using the well-known polynomial-time algorithms of [Kha80, Kar84]. Having the fractional values of the variables, we round each variable to be 1 with probability proportional to its fractional value.

When using this program for 2-spanning E^-, we get a good result; i.e., the randomized process gives a 2-spanner whose maximum degree is "close" to the "fractional degree" of the fractional solution. However, using this method we are not expected to 2-span all the edges of E^+. To see this, consider an edge e lying in $\Omega(n)$ triangles. The fractional program may give all (the variables of) the edges in these triangles a

value in $\Theta(1/n)$. In this way constraints (2) and (3) are easily satisfied. However, the probability of a triangle to "survive the randomized rounding" (i.e., to have both its edges set to 1) is $\Theta(1/n^2)$. Since there are only $\Theta(n)$ triangles, the edge is not expected to be 2-spanned. (This phenomenon captures the unfortunate "quadratic" behavior of our linear program.)

We therefore 2-span the edges of E^+ using a different procedure. We draw every edge $e \in E$ randomly to the spanner, with some fixed (small) probability. We then show that the edges of E^+, i.e., edges that belong to sufficiently many triangles, are likely to be 2-spanned in the resulting subgraph (namely are likely to lie on a triangle whose two other edges were selected by the randomized choice). We also show that with high probability the degree that is added to each vertex in this procedure is "small."

In the remainder of this section we present our approximation algorithm, and in the next section we give its analysis. Throughout the algorithm, we denote by E_u the set of edges yet to be 2-spanned.

ALGORITHM 5.1.

Input: A graph $G = (V, E)$ of maximum degree Δ.

1. *Let*
$$p = \frac{2 \cdot \sqrt{\log n}}{\Delta^{1/4}}, \qquad M = 2 \cdot \Delta^{1/4} \cdot \sqrt{\log n}$$
and set $E_r^1, E_r^2 \leftarrow \emptyset$.
2. *For every edge $e \in E$, draw e randomly and independently to be in the spanner with probability p. Let E_r^1 denote the set of edges selected into the spanner by the randomized process.*
3. *Set $E_u = \{e \in E \mid e \notin E_r^1$ and no two edges $e_1, e_2 \in E_r^1$ form a triangle with $e\}$.*
4. *Solve the fractional relaxation of the program $(P1)$ corresponding to E_u, E_r^1, and Δ^*.*
5. *Let $\{x_{i,j}, y_{i,k}\}$ be the optimal (fractional) solutions corresponding to $(P1)$. For every variable $\hat{x}_{i,j}$ create a respective random variable $\bar{x}_{i,j}$.*
6. *Randomly and uniformly set $\bar{x}_{i,j}$ to be 1 with probability $\min\{1, M \cdot x_{i,j}\}$.*
7. *If $\bar{x}_{i,j}$ is set to 1, then add the edge (v_i, v_j) to E_r^2.*
8. *Let E_u be the set of edges that are still unspanned by $E_r^1 \cup E_r^2$. Set $E_r = E_r^1 \cup E_r^2 \cup E_u$.*
9. *Output E_r.*

6. Analysis. First we explain how to overcome the assumption that Δ^* is known. Let Δ_f^* be the smallest value for which $(P1)$ has a feasible (fractional) solution. We call Δ_f^* the smallest *fractional* degree of the best *fractional* 2-spanner. Indeed, we only have to know (and run $(P1)$ with) Δ_f^* for our scheme to work. Clearly, $\Delta^* \geq \Delta_f^*$. This follows from the following simple claim.

LEMMA 6.1. *If we run the program* (P1) *with L replacing Δ^*, and* (P1) *has no fractional feasible solution, then $\Delta^* > L$.* □

The value Δ_f^* is found through binary search, by running $(P1)$ with values taken from the (discrete) interval $[\lceil\sqrt{\Delta}\,\rceil, \Delta]$. The search ends with some specific L such that the program succeed with $L+1$ but fails with L. By Lemma 6.1, $\Delta^* \geq \Delta_f^* > L$. On the other hand, we have a fractional feasible solution for $L+1$. Thus, we found the best fractional Δ_f^* (up to a difference of 1). For proving the desired approximation ratio, we show how to construct an (integer) spanner with maximum degree "close" to $L+1$ and therefore close to $\Delta^* (> L)$.

We now observe that the output is indeed a 2-spanner: the set of edges E_r forms a 2-spanner of G, since every edge not in E_r lies on a triangle with two edges of E_r.

Denote the optimum low-degree 2-spanner for G by H^*. Let us now proceed to bound from above the ratio between $\Delta(E_r)$ and $\Delta(H^*)$.

Throughout the subsequent analysis, we set $p = 2\sqrt{\log n}/\Delta^{1/4}$ and $M = 4\Delta^{1/4} \cdot \sqrt{\log n}$.

6.1. Handling E^+. Our first aim is to show that edges in E^+, i.e., edges with large $|Tri(e)|$, are likely to be 2-spanned in step 2 of our algorithm.

LEMMA 6.2. *With probability at least $1 - 1/n^2$, every edge $e \in E^+$ is 2-spanned in step 2 of Algorithm 5.1.*

Proof. Denote $m = |Tri(e)|$ and assume that $m \geq \sqrt{\Delta}$. Let $Tri(e) = \{T_1, T_2, \ldots, T_m\}$ with $T_i = \{e_1^i, e_2^i, e\}$; i.e., the three edges of T_i form a triangle in G. The probability that a triangle T_i does not 2-span e, namely, that neither e_1^i nor e_2^i are selected into the spanner in step 2, is $1 - p^2$. The probability that neither of the triangles 2-span e is

$$(1-p^2)^m \leq (1-p^2)^{\sqrt{\Delta}} = \left(\left(1-p^2\right)^{1/p^2}\right)^{4\log n} < \frac{1}{n^4}.$$

(The last inequality follows from the fact that $(1-x)^{1/x} \leq 1/e$ for $x \leq 1$.) Therefore, the probability that there exists one such an edge which is not 2-spanned is bounded by

$$\frac{|E|}{n^4} \leq \frac{1}{n^2}. \qquad \square$$

Next we estimate the maximum degree $\Delta(E_r^1)$ in the graph induced by E_r^1 by proving the following lemma.

LEMMA 6.3. *With probability at least $1 - 1/n^2$, $\Delta(E_r^1) \leq 4 \cdot \Delta^{3/4} \cdot \sqrt{\log n}$.*

Proof. For vertices v such that $\deg(v) < \Delta^{3/4} \cdot \sqrt{\log n}$, the claim follows vacuously. Hence we need to prove a degree bound only for vertices v such that $\deg(v) > \Delta^{3/4} \cdot \sqrt{\log n}$. Let $sp_1(e)$ be the random variable indicating if e was drawn to be in E_r^1; namely,

$$sp_1(e) = \begin{cases} 1, & e \in E_r^1, \\ 0 & \text{otherwise.} \end{cases}$$

Let $d_r^1(v)$ denote the random variable that equals the degree of v in the graph induced by E_r^1. Thus,

$$d_r^1(v) = \sum_{e \in E(v)} sp_1(e).$$

Clearly, $\mathbb{E}(sp_1(e)) = p$ and therefore

$$\mathbb{E}(d_r^1(v)) = \sum_{e \in E(v)} \mathbb{E}(sp_1(e)) = p \cdot \deg(v).$$

(Recall the assumption that $d(v) \geq \sqrt{\log n}\Delta^{3/4}$.) By the definition of p we get that the expected degree of v, $\mu(v)$ is bounded below by $2\log n\sqrt{\Delta}$. Since $d_r^1(v)$ is a sum

of independent Bernoulli trials, we can apply Lemma 2.2 to it with $\delta = 1$ and deduce that

$$\mathbb{P}\left(d_r^1(v) > 2 \cdot \mu(v)\right) \leq \left(\frac{e}{4}\right)^{\mu(v)} < \frac{1}{n^3}.$$

(For the last inequality, recall that one may assume that $\Delta = \Omega(\log^2 n)$. Indeed, here we only need Δ to be bounded below by some constant.)

Therefore, with probability at least $1 - 1/n^3$,

$$d_r^1(v) \leq 4 \cdot \Delta^{3/4} \cdot \sqrt{\log n}.$$

By summing up the probabilities over all the vertices, the lemma follows. □

(We note that as in [Rag88] slightly better results are attainable; i.e., it is possible to show a degree bound of $2 \cdot \sqrt{\log n}\Delta^{3/4} + o(\Delta^{3/4})$. However, we give the simpler bound here, since in our case we already have an approximation ratio of $\Delta^{1/4}$ and therefore the improvement can only (slightly) affect the constants. A similar situation holds, later, with regards to Lemma 6.4.)

6.2. Handling E^-. We would next like to show that the maximum degree in the subgraph selected by our linear program algorithm is small. Define the variable $d_r^2(v)$ to be the degree of v in the random choices made in steps 6 and 7 of Algorithm 5.1.

LEMMA 6.4. *With probability at least* $1 - 1/n^2$, $d_r^2(v) < 2M \cdot \Delta^*$ *for every* $v \in V$.

Proof. In order to establish an upper bound on the maximum degree, we must prove that in the random choices of steps 6 and 7 of Algorithm 5.1, the expected number of edges chosen for every vertex is "small." Let $v_i \in V$ be an arbitrary vertex, and let $u_1, \ldots, u_m$ be its neighbors. Let $e_j = (v_i, u_j), j = 1, \ldots, m$. We denote by $sp_2(e_j)$ the random indicator variable for the inclusion of e_j in the spanner, in steps 6 and 7; namely,

$$sp_2(e_j) = \begin{cases} 1, & e_j \text{ is chosen to be in the spanner in steps 6 and 7,} \\ 0 & \text{otherwise} \end{cases}$$

and thus

$$d_r^2(v_i) = \sum_{i=1}^{m} sp_2(e_j).$$

The expected value of $d_r^2(v_i)$ satisfies

$$\mathbb{E}(d_r^2(v_i)) = \sum_{j=1}^{m} \mathbb{E}(sp_2(e_j)) = \sum_{j=1}^{m} \min\{1, Mx_{ij}\} \leq M \cdot \sum_{j=1}^{m} x_{i,j}.$$

By (2) we have

$$\mathbb{E}(d_r^2(v_i)) \leq M \cdot \Delta^*.$$

Since $d_r^2(v_i)$ is the sum of independent Bernoulli trials, it follows from Lemma 2.2 with $\delta = 1$ that

$$\mathbb{P}\left(d_r^2(v_i) > 2 \cdot M \cdot \Delta^*\right) < \left(\frac{e}{4}\right)^{M\Delta^*} \leq \frac{1}{n^3}.$$

(The last inequality follows from the assumption that $\Delta \geq \Omega(\log^2 n)$ and from the definition of M.) Summing up the probabilities for all the vertices, the claim follows. ∎

In order to bound the number of edges added to each vertex v in step 8 of the algorithm (i.e., when E_u is added to E_r), we have to estimate how many edges are 2-spanned in steps 6 and 7. We show that with high probability, E_u is empty after step 7, and therefore E_u does not change $\Delta(E_r)$.

LEMMA 6.5. *The probability that an edge $e_k = (v_l, v_t)$ of E^- is 2-spanned in steps 6 and 7 is at least $1 - 1/n^4$.*

Proof. For the sake of proving the lemma we need the following technical lemma (cf. Chapter 8 of [AS92]).

LEMMA 6.6. *Let $\{A_i\}_{i=1}^m$ be m independent events and let $\mathbb{P}(A_i) = p_i$ and $\sum_{i=1}^m p_i \geq d$. Then $\mathbb{P}(\bigcup_{i=1}^m A_i) \geq 1 - 1/e^d$.* ∎

Let $e_k = (v_l, v_t) \in E^-$ and let $D(e) = \{v_1, \ldots, v_d\}$, $d = |Tri(e_k)| \leq \sqrt{\Delta}$. The inequality of the program $(P1)$ corresponding to e_k is

$$x_{l,t} + \sum_{v_i \in D(e_k)} y_{i,k} \geq 1.$$

Let A_i be the event that v_i 2-helps e_k in the chosen spanner. Let I_k be the event that e_k is included in the spanner. We now estimate the probability that A_i occurs. This probability equals the probability that both (v_i, v_l) and (v_i, v_t) are selected to the spanner. Clearly, we have only to consider the case that $\min\{M \cdot x_{i,t}, M \cdot x_{i,l}\} < 1$, for if this is not the case, $\mathbb{P}(A_i) = 1$. In the case both $M \cdot x_{i,t} < 1$ and $M \cdot x_{i,l} < 1$, $\mathbb{P}(A_i) = M^2 \cdot x_{i,t} \cdot x_{i,l}$. In the case that, say $M \cdot x_{i,t} \geq 1$ and $M \cdot x_{i,l} < 1$, we have $\mathbb{P}(A_i) = M \cdot x_{i,l} > M^2 \cdot x_{i,l}^2$. In either case, by (4) of the linear program we have

$$\mathbb{P}(A_i) \geq M^2 \cdot y_{i,k}^2. \tag{7}$$

The event $Cov(e_k) =$"e_k is 2-spanned" is the union $Cov(e_k) = \bigcup_i A_i \cup I_k$; namely, e_k is 2-spanned iff it is in the spanner or is 2-helped by some vertex. We now estimate the sum of probabilities of the A_i and I_k.

CLAIM 6.7. $\mathbb{P}(I_k) + \sum_{i=1}^d \mathbb{P}(A_i) \geq 3 \cdot \log n$.

Proof. We may assume that $x_{l,t} < 1/M$ since otherwise the edge e_k is taken into the spanner with probability 1 and hence is 2-spanned. We therefore have $\sum_{v_i \in D(e_k)} y_{i,k} > 1 - 1/M$.

We now have by (7)

$$\sum_{i=1}^d \mathbb{P}(A_i) \geq \sum_{i=1}^d M^2 y_{i,k}^2 = M^2 \sum_{i=1}^d y_{i,k}^2.$$

By the Cauchy–Schwartz inequality (cf. [Fla85]) we have

$$\begin{aligned}\sum_{i=1}^d \mathbb{P}(A_i) &\geq M^2 \cdot \frac{\left(\sum_{i=1}^d y_{i,k}\right)^2}{d} \geq M^2 \frac{(\sum_{i=1}^d y_{i,k})^2}{\sqrt{\Delta}} = 4 \log n \left(\sum_{i=1}^d y_{i,k}\right)^2 \\ &\geq 4 \log n (1 - 1/M)^2 > 3 \log n.\end{aligned}$$

(The last inequality holds for $n \geq 2$, in which case $M \geq 4$.) Therefore, this proves our claim. ∎

Note that the events I_k, A_i are all *independent*, since I_k and A_i all *concern different edges*. Thus we may apply Lemma 6.6 and get

$$\mathbb{P}(Cov(e_k)) \geq 1 - \frac{1}{e^{3\log n}} > 1 - \frac{1}{n^4},$$

proving the lemma. □

Thus, with probability at least $1 - 1/n^2$, all the edges are 2-spanned in step 7. Therefore, with probability at least $1 - 1/n^2$, $E_u = \emptyset$, in which case E_u does not increase the maximum degree. In summary, Lemmas 6.2, 6.3, 6.4 and 3.2, combined with the above discussion, yield the following theorem.

THEOREM 6.8. *With probability at least $1 - 1/n$, the algorithm produces a 2-spanner with maximum degree bounded by $O(\sqrt{\log n}\Delta^{1/4}\Delta^*)$.* □

COROLLARY 6.9. *Algorithm* 5.1 *produces a 2-spanner that with probability at least $1 - 1/n$ is an $O(\sqrt{\log n} \cdot \Delta^{1/4})$ approximation for the* LD-2SP *problem.* □

Note that the error probability can be reduced to $1/n^c$ for any (constant) c, losing only constants in the approximation ratio.

7. Derandomization. In this section, we show how to transform our randomized algorithm into a deterministic one. We use the well-known "method of conditional probabilities" (cf. [Spe87]) and its generalization, the method of "pessimistic estimators" [Rag88].

7.1. The method of pessimistic estimators. Let us first describe the method of pessimistic estimators in a form which is convenient for our purpose. Let $q_1, \ldots, q_l$ be random Boolean variables, set to 0 or 1 with some probabilities, and consider the probability space $\mathcal{Q} = \{(q_1, \ldots, q_l) \mid q_i \in \{0,1\}, 1 \leq i \leq l\}$ of 2^l points. Let $X_1, \ldots, X_s$ be a collection of "bad" events over $\mathcal{Q}$, and suppose that $\mathbb{P}(X_i) = p_i$ and that $\sum_{i=1}^s p_i < 1$. Thus the event $\bigcap_i \bar{X}_i$ has positive probability. We therefore have a point $(\hat{q}_1, \ldots, \hat{q}_l)$ in the probability space $\mathcal{Q}$ for which $\bigcap_i \bar{X}_i$ holds. Suppose that for each event X_i and for each $0 \leq j \leq l$ we have a function $f_j^i(q_1, \ldots, q_j)$ for which the following holds.

(1) $\sum_{i=1}^s f_{j-1}^i(q_1, \ldots, q_{j-1}) \geq \min\{\sum_{i=1}^s f_j^i(q_1, \ldots, q_{j-1}, 0), \sum_{i=1}^s f_j^i(q_1, \ldots, q_{j-1}, 1)\}$ for all $1 \leq i \leq s$, $0 \leq j \leq l$.
(2) $f_j^i(q_1, \ldots, q_j) \geq \mathbb{P}(X_i \mid q_1, \ldots, q_j)$.
(3) $\sum_{i=1}^s f_0^i < 1$.
(4) The function $f_j^i(q_1, \ldots, q_j)$ can be computed in polynomial time in l for every i and j and $(q_1, \ldots, q_j) \in \{0,1\}^j$. Also, the number of events, s, is polynomial in l.

In this case one can transform the probabilistic existence proof into a polynomial algorithm (in terms of l). This is done by fixing the value of q_i to be 0 or 1 iteratively, one by one. In the jth step, having determined the values of $q_1, \ldots, q_{j-1}$, we decide upon the value q_j (setting it either to 0 or to 1) so as to minimize the sum $\sum_{i=1}^s f_j^i(q_1, \ldots, q_j)$. It easily follows from the above conditions that the sum $\sum_{i=1}^s f_j^i(q_1, \ldots, q_j)$ never increases and, consequently (it follows from properties 2 and 3 that), at the end of the procedure we remain with a point $(\hat{q}_1, \ldots, \hat{q}_l)$ in the sample space for which the event $\bigcap_i \bar{X}_i$ holds. Also by property (4) above, this derandomization procedure can be executed in time polynomial in l.

The functions f_j^i are called *pessimistic estimators* for the actual conditional probabilities. The method of conditional probabilities is the special case where $f_j^i(q_1, \ldots, q_j) = \mathbb{P}(X_i \mid q_1, \ldots, q_j)$ and property 3 holds, i.e., the case where in addition to property 3

the conditional probabilities can be computed efficiently, so no estimators are needed. When this is the case, the remaining properties 1 and 2 follow immediately.

7.2. Derandomizing the 2-spanner algorithm. In the case of the 2-*spanner* problem we have a two-stage randomized procedure. Let us first focus on the harder task of derandomizing the second stage, where we 2-span the edges of E^- using randomized rounding. We draw the edges with different probabilities (that depend upon the values of the corresponding variables in the linear program). Our q_i variables, therefore, correspond to the edges, where every edge has some probability to be 1. We will identify the edge e_k with its corresponding random variable. (We will therefore say "e_k was set to 1" meaning that e_k was added to the spanner.)

We now describe the "bad events" in the second stage. The event $\mathcal{D}(v_i)$ is the event where the degree of v_i is greater than $2 \cdot M \cdot \Delta^*$. The event $\mathcal{U}(e_j)$ is the event that an edge $e_j \in E^-$ is unspanned at the end of Algorithm 5.1.

Throughout the sequel, we assume that we have already decided upon the values of $e_1, \ldots, e_{j-1}$, setting them either to 1 or 0, and we want to decide the value of e_j. We denote by $p(e_k)$ the probability by which e_k is drawn in the randomized rounding. (This probability equals M times the value of e_k in the linear program.) We use the following notation (that depends upon previous decisions). The number $\tilde{e}_k$ is defined as

$$\tilde{e}_k = \begin{cases} 1, & e_k \text{ was previously set to 1,} \\ 0, & e_k \text{ was previously set to 0,} \\ p(e_k), & \text{the value of } e_k \text{ was not yet determined.} \end{cases}$$

We next define the pessimistic estimator for the event $\mathcal{U}(e_j)$. Given some edge $e_k = (v_s, v_t)$ say that e_k *lies on* $t_k = |Tri(e_k)|$ *triangles* and denote $Tri(e_k) = \{(e_i^1, e_i^2, e_k)\}_{i=1}^{t_k}$. We then set the pessimistic estimators for $\mathcal{U}(e_k)$ to be

$$h_j^k(e_1, \ldots, e_j) = \Pi_{i=1}^{t_k}(1 - \tilde{e}_i^1 \cdot \tilde{e}_i^2).$$

We note that the above expression, *exactly* equals the probability that e_k is unspanned by either of its triangles (given the previous decisions). Note that e_k may be 2-spanned also if e_k itself is drawn into the spanner. This observation proves property 2 for the functions $\mathcal{U}(e_k)$ and $h_j^k(e_1, \ldots, e_j)$. Property 4 follows trivially.

We now turn our attention to the event $\mathcal{D}(v_i)$, meaning that the degree of v_i exceeds $2 \cdot M \cdot \Delta^*$. For these events, the conditional probabilities can be calculated in polynomial time using *dynamic programming*. However, this is relatively time consuming, as $O(d(v)\Delta^*)$ time is required in order to calculate each conditional probability associated with every vertex. It is therefore convenient to introduce the following pessimistic estimators, which are a special case of some estimators introduced in [Rag88]. Suppose that the edges of v_i are $e_1^i, e_2^i, \ldots, e_{d_i}^i$ (where d_i is the degree of v_i). For $\mathcal{D}(v_i)$ define the following pessimistic estimator. Set

$$g_j^i(e_1, \ldots, e_j) = \frac{\Pi_{r=1}^{d_i}\left(\tilde{e}_r^i + 1\right)}{4^{M \cdot \Delta^*}}.$$

The required property 2 follows in a way similar to the proof of the Chernoff bound, as in [Rag88]. Property 4 also follows trivially.

We now prove property 3. Note that since (as in [Rag88])

$$g_0^i = \frac{\Pi_{k=1}^{d_i}(p(e_k)+1)}{4^{M\cdot\Delta^*}} \le \frac{\Pi_{k=1}^{d_i} e^{p(e_k)}}{4^{M\cdot\Delta^*}} = \frac{e^{\sum_{k=1}^{d_i} p(e_k)}}{4^{M\cdot\Delta^*}} \le \left(\frac{e}{4}\right)^{M\cdot\Delta^*} \le \frac{1}{n^3},$$

property 3 follows from Lemmas 6.4 and 6.5. (In fact, the sum of the pessimistic estimators of the bad events is not only smaller than 1 but is also smaller than $1/n$. Nevertheless, the degrees of the vertices can only be reduced by a constant factor.)

Finally, we have to check property 1. We have

$$S = \sum_{i=1}^{n} g_{j-1}^i + \sum_{k=1}^{m} h_{j-1}^k = \sum_{i=1}^{n} \frac{\Pi_{r=1}^{d_i}\left(\tilde{e}_r^i + 1\right)}{4^{M\cdot\Delta^*}} + \sum_{k=1}^{m} \Pi_{i=1}^{t_k}(1 - \tilde{e}_i^1 \cdot \tilde{e}_i^2).$$

We can write S as the sum $S = S_1(e_j) + S_2(e_j) + S_3$ where

$$S_1(e_j) = (1 + p(e_j)) \cdot \Pi_1, \quad S_2(e_j) = \sum_l (1 - p(e_j) \cdot \tilde{e}_l) \cdot \Pi_2^l,$$

and the expressions Π_1, Π_2^l, and S_3 do not contain $\tilde{e}_j$. When setting e_j to 1, the difference between the terms in the first summand is $S_1(1) - S_1(e_j) = (1 - p(e_j)) \cdot \Pi_1$, and the difference in the second summand is $S_2(1) - S_2(e_j) = -\sum_l \tilde{e}_l(1 - p(e_j)) \cdot \Pi_2^l$. So if $\sum_l \tilde{e}_l \cdot \Pi_2^l \ge \Pi_1$ we are done. Otherwise, when setting e_j to 0, the difference in the first summand is $S_1(0) - S_1(e_j) = -p(e_j) \cdot \Pi_1$ and in the second summand, $S_2(0) - S_2(e_j) = \sum_l p(e_j) \cdot \tilde{e}_l \cdot \Pi_2^l < p(e_j) \cdot \Pi_1$. Thus the required property follows.

Now we have to consider the first (and easier to derandomize) stage of the algorithm, where we handle the edges of E^+. In the first stage, we draw all the edges independently and uniformly with probability $p = 2 \cdot \sqrt{\log n}/\Delta^{1/4}$. The bad event here is similar. We have the event $\mathcal{D}(v_i)$ which is the bad event that the degree of v_i exceeds $4 \cdot \Delta^{3/4} \cdot \sqrt{\log n}$. The event $\mathcal{U}(e_k)$ is the event that an edge $e \in E^+$ is not 2-spanned at the end of Algorithm 5.1.

Thus, the derandomization of this first stage is a special case of the derandomization of the second stage.

We have therefore established the following result.

COROLLARY 7.1. *Algorithm* 5.1*, together with a derandomization procedure, produces a* 2*-spanner that is an* $O(\sqrt{\log n} \cdot \Delta^{1/4})$ *approximation for the* LD-2SP *problem.* □

8. An algorithm for sparse graphs. In this section we present a relatively simple algorithm $Sparse_1$ that performs better than Algorithm 5.1 in the case where the underlying graph is sparse. In this case, algorithm $Sparse_1$ yields a $2\sqrt{E}$ *additive* approximation. (By "α additive approximation" we mean that the resulting degree is $\Delta^* + \alpha$.) Thus, if the number of edges is up to $n^{3/2}$, we get an additive term of less than $n^{3/4}$ (or, alternatively, a very low multiplicative factor). For the range $n^{3/2} \le E \le n^{7/4}$ we have a different algorithm $Sparse_2$ that slightly improves Algorithm 5.1 in the worst case. This algorithm is considerably more complicated and is therefore omitted. The interested reader is referred to [KP93].

8.1. Algorithm $Sparse_1$. Algorithm $Sparse_1$ divides the vertex set into "heavy" and "light" vertices. The set $Heavy$ consists of vertices with degrees *at least* $\sqrt{E}$, and

the set *Light* consists of vertices whose degrees are *less than* $\sqrt{E}$. Then the set of all edges with both endpoints in $Heavy$, $E(Heavy)$, and all edges with both endpoints in Light, $E(Light)$, are taken into the spanner. The cut edges, with one endpoint in $Heavy$ and the other in *Light*, are 2-spanned using a linear programming formulation. The output edge set is denoted E_r. Again we may assume that Δ^* (or, more accurately, Δ^*_f) is known in advance.

ALGORITHM 8.1. *Algorithm* $Sparse_1$
Input: A graph $G = (V, E)$.

1. *Let* $Heavy = \{v \in V \mid \deg(v) \geq \sqrt{E}\}$, $\quad Light = \{v \in V \mid \deg(v) < \sqrt{E}\}$.
2. *Add* $E(Heavy) \cup E(Light)$ *into* E_r.
3. *Let* E_u *be the set of cut edges having one endpoint in Heavy and one in Light.*
4. *Solve the fractional relaxation of the program* $(P1)$ *corresponding to* E_u, E_r, *and* Δ^*.
5. *Let* $\{x_{i,j}, y_{i,k}\}$ *be the optimum (fractional) solutions corresponding to* $(P1)$. *For every variable* $\hat{x}_{i,j}$ *create a respective random variable* $\bar{x}_{i,j}$.
6. *Randomly and uniformly set* $\bar{x}_{i,j}$ *to be* 1 *with probability* $\min\{1, 4 \log n \cdot x_{i,j}\}$.
7. *If* $\bar{x}_{i,j}$ *is set to* 1, *then add the edge* (v_i, v_j) *to* E_r.
8. *Add all the remaining non* 2*-spanned edges to* E_r *and output* E_r.

8.2. Analysis. We now prove that Algorithm 8.1 yields a $2\sqrt{E}$ additive approximation. We first note the following simple fact.

FACT 8.2. *In step* 2 *of Algorithm* 8.1, *we add to* E_r *no more than* $2\sqrt{E}$ *edges adjacent to any vertex.*

Proof. The claim is clear for vertices v in *Light*, since such a vertex has at most $\sqrt{E}$ adjacent edges. Also note that $|Heavy| \leq 2 \cdot \sqrt{E}$, and thus for a vertex $v \in Heavy$ at most $2 \cdot \sqrt{E}$ edges are candidates for addition to E_r in this step. The claim follows. □

We now note the following simple yet crucial fact.

FACT 8.3. *In every triangle corresponding to a cut edge* e *in the set* E_u *defined in step* 3 *of Algorithm* 5.1, *exactly one of its edges was already added to* E_r *in Step* 2.

Proof. Every such triangle contains an edge e' that is not a cut edge. Thus, either both vertices of e' belong to $Heavy$ or they both belong to *Light*. In either case e' was added to E_r in step 2. □

Given some cut edge $e_k = (v_l, v_t)$, let $y_{i,k}$ be a variable corresponding to $v_i \in D(e_k)$ and e_k. Without loss of generality, let (v_i, v_l) be the other cut edge in the triangle. Thus, the probability that v_i 2-helps e_k exactly equals $\min\{1, 4 \log n \cdot x_{i,l}\} \geq \min\{1, 4 \log n \cdot y_{i,k}\}$. Thus a proof along the lines of that in Lemma 6.5 (but simpler, due to the fact that here we can avoid the "squaring" effect) gives the next corollary. The main point is that the sum of the probabilities that the triangles of e_k survive is roughly $4 \log n \sum_i y_{ik} = \Omega(\log n)$. (The squaring effect is avoided, since in any triangle we need only one edge to survive and not two edges together, because the other edge of every triangle was already chosen to the spanner.)

COROLLARY 8.4. *With probability at least* $1 - 1/n^2$ *all the edges in* E_u *are* 2*-spanned by the end of step* 7 *of Algorithm* 8.1. □

We note that the expectation of the degree of a vertex (and thus, using Lemma 2.2, the expectation of the maximum degree) is bounded by $2\sqrt{E} + O(\log n \cdot \Delta^*)$. Thus up to logarithmic factors, this approximation is $2\sqrt{E}$ additive. By using derandomization (as explained in the previous section) and Lemma 3.2 we have the following.

COROLLARY 8.5. *There is an* $\tilde{O}(\sqrt{E/\Delta})$ *approximation algorithm for the* LD-2SP *problem.* □

Note that if $E \leq \Delta\sqrt{\Delta}$ then Algorithm 8.1 performs better than Algorithm 5.1. Let us consider the approximation ratio in terms of n. Algorithm 5.1 gives a worst-case ratio of $\tilde{O}(n^{1/4})$. This happens when Δ is very large, i.e., $\Delta = \Theta(n)$. On the other hand, if Δ is very large, we may have $\sqrt{|E|/\Delta}$ small. This implies that the ratio is improved whenever $|E| < n^{3/2}$ by doing the following. Considering a graph with $n^{3/2-\epsilon}$ edges, if $\Delta \leq n^{1-2\epsilon/3}$ then apply Algorithm 5.1, else apply Algorithm 8.1. This gives the following bound.

COROLLARY 8.6. *For every* $0 \leq \epsilon \leq 1/2$, *there exists an* $\tilde{O}(n^{1/4-\epsilon/6})$ *approximation algorithm for the* LD-2SP *problem on graphs* G *with* $E = O(n^{3/2-\epsilon})$. □

This result improves the ratio in "absolute terms" (i.e., in terms of n). For example, if $E = O(n)$ then the combined algorithm has an $O(n^{1/6})$ approximation ratio (whereas Algorithm 5.1 would give an $O(n^{1/4})$ ratio in the worst case).

8.3. The case of $n^{3/2} < E \leq n^{7/4}$. In [KP93] we present Algorithm 8.1 which outperforms Algorithm 5.1 in the range $n^{3/2} < E \leq n^{7/4}$. That is, we assume a graph with $O(n^{7/4-\epsilon})$ edges, where $0 \leq \epsilon \leq 1/4$, and show an $\tilde{O}(n^{1/4-\epsilon/11})$-ratio approximation algorithm, improving over the $n^{1/4}$-ratio of Algorithm 5.1.

THEOREM 8.7 (see [KP93]). *Given a graph* $G = (V, E)$ *with* $n^{7/4-\epsilon}$ *edges, Algorithm* 8.1 *combined with a derandomization procedure has an* $\tilde{O}(n^{1/4-\epsilon/11})$ *approximation ratio.* □

9. Spanning the edges of a single vertex. In this section we consider the weaker problem of spanning the edges adjacent to a single vertex and present an $O(\log n)$ approximation for it. This construction can easily be applied to span the edges adjacent to a small subset of the vertices.

Say that we are given a specific vertex v (presumably with high degree) and we want to 2-span its edges (and do not care about the edges not touching v). Thus our aim is to select some subset $E' \subseteq E$ inducing low degrees such that every missing edge of v is 2-spanned by a triangle. Denote this problem by SLD-2SP.

First we note that the lower bound on approximability for the LD-2SP problem applies to the SLD-2SP problem as well. Given an instance of the set cover problem, we may construct $\bar{G}$ as in section 4, and consider the problem of spanning the edges of s. A similar proof as in section 4 shows that unless $NP \subset DTIME(n^{\log\log n})$, the ratio of any approximation algorithm for the SLD-2SP problem is no better than $\ln n/5$. On the other hand, in this section we match this result, showing a logarithmic-ratio approximation algorithm for SLD-2SP. We use a known greedy approximation for a (slightly) more involved version of the set cover problem. This essentially shows that this weaker problem SLD-2SP is equivalent to set cover, with respect to approximation.

The *bounded-load* set cover problem is a variant of the set cover problem that deals with assigning specific covering vertices to the covered vertices. Namely, along with finding a cover C of V_2, it is required to provide a function $\varphi : V_2 \to C$, assigning each vertex v_2 in V_2 a neighbor $\varphi(v_2)$ in C. The load of a vertex $v_1 \in V_1$ is defined as the number of covered vertices it is assigned to, i.e., $L(v_1) = |\{v_2 \in V_2 \mid \varphi(v_2) = v_1\}|$. The problem is now defined as follows. Given a bipartite graph $G(V_1, V_2, E)$ and an integer $L \leq |V_2|$, find a cover $C \subset V_1$ of V_2 and an assignment φ with maximum load bounded by L (i.e., such that no vertex in V_1 is assigned to more than L vertices of V_2).

We recall the following theorem of [Wol82].

THEOREM 9.1 (see [W82]). *The bounded load set cover problem can be approximated with ratio* $O(\log |V_2|)$. □

Given an instance of the SLD-2SP problem, where our aim is to 2-span the edges of v, we reduce it to an instance of the bounded load set cover problem. We use a reformulation of the SLD-2SP problem as follows. Model the neighbors $N(v)$ of v and the edges $E(v)$ of v in a bipartite graph $Bip = (N(v), E(v), A)$ where a vertex $u \in N(v)$ is connected to an edge $e = (v, w) \in E(v)$, iff $(u, w) \in E$. (Namely, $(u, e) \in A$ iff u belongs to $D(e)$ and can 2-help $e = (v, w) \in E(G)$ in the spanner by the edges (u, w) and (u, v).)

The aim is to find a *small-sized* cover C of $E(v)$ (in Bip), with *small* maximum load. The merit of this construction is explained by the following observation.

CLAIM 9.2. *Given a small set C covering $E(v)$ with maximum load L, it is possible to construct a 2-spanner of the edges of v with maximum degree bounded by* $\max\{|C|, L + 1\}$.

Proof. Construct the 2-spanner as follows. Define the bipartite graph Bip, and let $C = \{w_1, \ldots, w_k\} \subseteq N(v)$. Let $\{e^i_1, \ldots, e^i_{n_i}\}$ be the edges incident to v that are covered in Bip by w_i (where $n_i \leq L$). Let $e^i_j = (v, z^i_j)$. Add the edges $\{(v, w_i)\}_{i=1}^k$ to the spanner. Also, add the edges (w_i, z^i_j). Clearly, we have added $|C|$ edges adjacent to v (since one edge is added to v for each vertex of C). We have also added no more than $L + 1$ edges adjacent to w_i for every i, since w_i covers no more than L edges $e_j = (w_i, z^i_j)$ in the spanner and also the edge (v, w_i). Thus the SLD-2SP problem is equivalent to finding a cover C and some assignment with load L, minimizing $\max\{|C|, L + 1\}$. □

It easily follows from Theorem 9.1 that the problem of finding a cover with a load assignment minimizing $\max\{|C|, L + 1\}$ also enjoys a logarithmic approximation. In turn, this gives a logarithmic approximation ratio for the problem of spanning the edges of v.

COROLLARY 9.3. *The problem of spanning the edges of v with low degree has a polynomial time approximation algorithm with ratio* $O(\log n)$. *Conversely, the problem cannot be approximated with ratio better than* $\ln n/5$, *unless* $NP \subset DTIME(n^{\log \log n})$ *holds.* □

As an additional by-product, if we are required to 2-span *only* the edges of a collection of k of the graph vertices, for small k, we can 2-span the edges of every vertex in the set one by one and get an $O(k \log n)$ approximation for this problem.

COROLLARY 9.4. *The problem of 2-spanning the edges of a subset $V' \subset V$ of k vertices with minimum maximum degree can be approximated within an* $O(\log n \cdot k)$ *ratio.* □

REFERENCES

[ABP91] B. AWERBUCH, A. BARATZ, AND D. PELEG, *Efficient Broadcast and Light-Weight Spanners*, 1991, manuscript.

[ADDJ90] I. ALTHÖFER, G. DAS, D. DOBKIN, AND D. JOSEPH, *Generating sparse spanners for weighted graphs*, in Proc. 2nd Scandinavian Workshop on Algorithm Theory, Vol. LNCS-447, Springer-Verlag, Berlin, New York, 1990, pp. 26–37.

[AS92] N. ALON AND J. SPENCER, *The Probabilistic Method*, John Wiley and Sons Inc., New York, 1992.

[BGLR93] M. BELLARE, S. GOLDWASSER, C. LUND, AND A. RUSSELL, *Efficient probabilistically checkable proofs: Applications to approximation*, in Proc. 25th ACM Symposium on the Theory of Computing, 1993, pp. 294–304.

[Cai91] L. CAI, *Tree 2-spanners*, Technical Report 91-4, Simon Fraser University, Burnaby, BC, 1991.

[CDNS92] B. Chandra, G. Das, G. Narasimhan, and J. Soares, *New sparseness results on graph spanners*, in Proc. 8th ACM Symposium on Computational Geometry, 1992.

[Che52] H. Chernoff, *A measure of asymptotic efficiency for tests of hypothesis based on the sum of observations*, Ann. Math. Statist., 23 (1952), pp. 493–507.

[Che86] L. P. Chew, *There is a planar graph almost as good as the complete graph*, in ACM Symposium on Computational Geometry, 1986, pp. 169–177.

[DFS87] D. P. Dobkin, S. J. Friedman, and K. J. Supowit, *Delaunay graphs are almost as good as complete graphs*, in Proc. 31st IEEE Symp. on Foundations of Computer Science, 1987, pp. 20–26.

[DJ89] G. Das and D. Joseph, *Which triangulation approximates the complete graph?* in International Symposium on Optimal Algorithms, Vol. LNCS-401, Springer-Verlag, Berlin, New York, 1989, pp. 168–192.

[Fei96] U. Feige, *A threshold of* $\ln n$ *for approximating set cover*, in Proc. ACM STOC, 1996.

[Fla85] H. Flanders, *Calculus*, W.H. Freeman & Co., San Francisco, CA, 1985.

[HW56] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*. Oxford University Press, London, 1956.

[Joh74] D. S. Johnson, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci., 9 (1974), pp. 256–278.

[Kar84] N. Karmarkar, *A new polynomial time algorithm for linear programming*, Combinatorica, 4 (1984), pp. 373–395.

[Kha80] L. G. Khachian, *Polynomial algorithms in linear programming*, Zh. Vychisl. Mat. i Mat. Fiz. (1980), pp. 53–72.

[KP93] G. Kortsarz and D. Peleg, *Generating Low-Degree* 2*-Spanners*, Technical Report CS93-07, The Weizmann Institute, Rehovot, Israel, 1993.

[KP94] G. Kortsarz and D. Peleg, *Generating sparse* 2*-spanners*, J. Algorithms, 17 (1994), pp. 222–236.

[LL89] C. Levcopoulos and A. Lingas, *There are planar graphs almost as good as the complete graph and as short as minimum spanning trees*, in International Symposium on Optimal Algorithms, Vol. LNCS-401, Springer-Verlag, New York, Berlin, 1989, pp. 9–13.

[Lov75] L. Lovász, *On the ratio of integral and fractional covers*, Discrete Math., 13 (1975), pp. 383–390.

[LR94] A. L. Liestman and D. Richards, *Degree-constrained pyramid spanners*, J. Parallel Distr. Comput., 25 (1995), pp. 1–6.

[LS93a] A. L. Liestman and T. C. Shermer, *Additive graph spanners*, Networks, 23 (1993), pp. 343–364.

[LS93b] A. L. Liestman and T. C. Shermer, *Grid spanners*, Networks, 23 (1993), pp. 123–133.

[LY93] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, in Proc. 25th ACM Symposium on the Theory of Computing, 1993, pp. 286–293, 1993.

[PS89] D. Peleg and A. Schäffer, *Graph spanners*, J. Graph Theory, 13 (1989), pp. 99–116.

[PU89] D. Peleg and J. D. Ullman, *An optimal synchronizer for the hypercube*, SIAM J. Comput., 18 (1989), pp. 740–747.

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

[RT87] P. Raghavan and C. D. Thompson, *Randomized rounding: A technique for provably good algorithms and algorithmic proofs*, Combinatorica, 7 (1987), pp. 365–374.

[Soa92] J. Soares, *Approximating Euclidean Distances by Small Degree Graphs*, Technical Report 92-05, University of Chicago, 1992.

[Spe87] J. Spencer, *Ten Lectures on the Probabilistic Method*, SIAM, Philadelphia, 1987.

[Wol82] L. A. Wolsey, *An analysis of the greedy algorithm for the submodular set covering problem*, Combinatorica, 2 (1982), pp. 385–393.

SIAM J. COMPUT.
Vol. 27, No. 5, pp. 1457–1491, October 1998
© 1998 Society for Industrial and Applied Mathematics
014

PERFORMING WORK EFFICIENTLY IN THE PRESENCE OF FAULTS*

CYNTHIA DWORK†, JOSEPH Y. HALPERN‡, AND ORLI WAARTS§

Abstract. We consider a system of t synchronous processes that communicate only by sending messages to one another, and together the processes must perform n independent units of work. Processes may fail by crashing; we want to guarantee that in every execution of the protocol in which at least one process survives, all n units of work will be performed. We consider three parameters: the number of messages sent, the total number of units of work performed (including multiplicities), and time. We present three protocols for solving the problem. All three are work optimal, doing $O(n+t)$ work. The first has moderate costs in the remaining two parameters, sends $O(t\sqrt{t})$ messages, and takes $O(n+t)$ time. This protocol can be easily modified to run in any completely asynchronous system equipped with a failure detection mechanism. The second sends only $O(t \log t)$ messages, but its running time is large ($O(t^2(n+t)2^{n+t})$). The third is essentially time optimal in the (usual) case in which there are no failures, and its time complexity degrades gracefully as the number of failures increases.

Key words. fault tolerance, work, Byzantine agreement, load balancing, distributed systems

AMS subject classifications. 68Q22, 68M10, 90B25

PII. S0097539793255527

1. Introduction.

A fundamental issue in distributed computing is fault tolerance: guaranteeing that work is performed, despite the presence of failures. For example, in controlling a nuclear reactor it may be crucial for a set of valves to be closed before fuel is added. Thus the procedure for verifying that the valves are closed must be highly fault tolerant. If processes never fail then the *work* of checking that the valves are closed could be distributed according to some load-balancing technique. Since processes may fail, we would like an algorithm that guarantees that the work will be performed as long as at least one process survives. Such an algorithm could be particularly useful in a local area network, where jobs might be distributed among idle workstations. (The idea of running computations on idle nodes is an old one, going back at least to [17]. See [12] for one implementation of this approach and for further references.) In this case a "failure" might correspond to a user reclaiming her machine.

The notion of work in this paper is very broad, but is restricted to "idempotent" operations, that is, operations that can be repeated without harm. This is because if a process performs a unit of work and fails before telling a second process of its achievement, then the second process has no choice but to repeat the given unit of

*Received by the editors September 14, 1993; accepted for publication (in revised form) August 15, 1996; published electronically May 19. A preliminary version of this work appeared in *Proc. 11th ACM Symposium on Principles of Distributed Computing*, 1992.

http://www.siam.org/journals/sicomp/27-5/25552.html

†IBM Research Division, Almaden Research Center, K53-B2, 650 Harry Road, San Jose, CA 95120-6099 (dwork@almaden.ibm.com).

‡Computer Science Department, Cornell University, Ithaca, NY 14853 (halpern@cs.cornell.edu). Most of the work of this author was performed while the author was at IBM Almaden Research Center.

§Computer Science Division, University of California, Berkeley, Berkeley, CA 94720 (waarts@cs.berkeley.edu). During this research, this author was at Stanford University and was supported by U.S. Army Research Office grant DAAL-03-91-G-0102, NSF grant CCR-8814921, ONR contract N00014-88-K-0166, and an IBM fellowship.

work. Examples include verifying a step in a formal proof, evaluating a boolean formula at a particular assignment to the variables, sensing the status of a valve, closing a valve, sending a message, say, to a process outside of the given system, or reading records in a distributed database.

Formally, we assume that we have a synchronous system of t processes that are subject to crash failures, that want to perform n independent units of work. (For now, we assume that initially there is common knowledge among the t processes about the n units of work to be performed. We return to this point later.) In one time unit a process can compute locally and perform one unit of work and one round of communication (sending and receiving messages). Given that performing a unit of work can be repeated without harm, a trivial solution is obtained by having each process perform every unit of work. In our original example, this would mean that every process checks that every valve is closed. This solution requires no messages, but in the worst case it performs tn units of work and runs in n rounds. (Here the worst case is when no process fails.)

Another straightforward solution can be obtained by having only one process performing the work at any time and checkpointing to each process after completing every unit of work. In this solution, at most $n+t-1$ units of work are ever performed, but the number of messages sent is almost tn in the worst case.

In both these solutions the total amount of *effort*, defined as work plus messages, is $O(tn)$. If the actual cost of performing a unit of work is comparable with the cost of sending a message, then neither solution is appealing. In this paper we focus on solutions that are work optimal, up to a constant factor, while keeping the total effort reasonable. Clearly, since a process can fail immediately after performing a unit of work, before reporting that unit to any other process, a work-optimal solution performs $n+t-1$ units of work in the worst case. Thus we are interested in solutions that perform $O(n+t)$ work.

Let $n' = \max(n, t)$. Our first result is an algorithm whose total effort is at most $3n' + 9t\sqrt{t}$. In fact, in the worst case the amount of work performed is at most $3n'$ and the number of messages is at most $9t\sqrt{t}$, so the form of the bound explains the costs exactly. We then optimize this algorithm to achieve running time of $O(n+t)$ rounds. Note that any solution requires n rounds in the worst case, since if $t-1$ processes are initially faulty then the remaining process must perform all n units of work. In this algorithm the synchrony is used only to detect failures, as usual by detecting the absence of an expected message. Thus it can be easily modified to work in a completely asynchronous system equipped with a failure detection mechanism.

We then prove that the above algorithm is not message optimal (among work-optimal algorithms), by constructing a technically challenging work-optimal algorithm that requires only $O(t \log t)$ messages in the worst case. Since $O(n+t)$ is a lower bound on work, and hence on effort, the $O(n+t \log t)$ effort of this algorithm is nearly optimal. The improved message complexity is obtained by a more aggressive use of synchrony. In particular, the absence of a message in this algorithm has two possible meanings: either the potential sender failed or it has insufficient "information" (generally about the history of the execution) and therefore has chosen not to send a message. Due to this use of synchrony, unlike the first algorithm, this low-effort algorithm will not run in the asynchronous model with failure detection. In addition, the efficiency comes at a price in terms of time: in the worst case, the algorithm requires $O(t^2(n+t)2^{n+t})$ rounds.

The first two algorithms are very sequential: at all times work is performed by a single active process that uses some checkpointing strategy to inform other processes about the completed work. This forces the algorithms to take at least n steps, even in a failure-free run. To reduce the time we need to increase parallelism. However, intuitively, increasing parallelism while simultaneously minimizing time and still remaining work optimal may increase communication costs, since processes must quickly tell each other about completed work. The third algorithm does exactly this in a fairly straightforward way, paying a price in messages in order to decrease best-case time. It is designed to perform time optimally in the absence of failures and to have its time complexity degrade gracefully with additional faults. In particular, it takes $n/t+2$ rounds in the failure-free case, and its message cost is $2t^2$; its worst-case message cost is $O(ft^2)$, where f is the actual number of failures in the execution.

There are a number of assumptions in our model that are arguably not realistic. For one, we assume that the n units of work are identical or, at least, that they all take the same amount of time to perform. In addition, we assume that the total workload is static and is common knowledge at the beginning of the algorithm. It is not too hard to modify our last algorithm to deal with a more realistic scenario, where work is continually coming in to different sites of the system, and is not initially common knowledge. We remark that a patent has been filed by IBM for such a modified algorithm.

One application of our algorithms is to Byzantine agreement. The idea is that the general tries to inform t processes, and then each of these t processes performs the "work" of ensuring that all processes are informed. In particular, our second algorithm, called Protocol $\mathcal{B}$, gives a Byzantine agreement algorithm for the crash fault model that requires $O(n+t\sqrt{t})$ messages and $O(n)$ time, where n is the number of processes in the system and t is a bound on the number of failures, while our third algorithm gives a Byzantine agreement algorithm that uses $O(n + t \log t)$ messages and exponential time. The best result prior to ours was a nonconstructive algorithm due to Bracha that requires $O(n + t\sqrt{t})$ messages [4]. Galil, Mayer, and Yung [11] have recently obtained an algorithm that uses only a linear number of messages. However, the algorithm is not comparable with the agreement algorithm obtained using Protocol $\mathcal{B}$ because it requires a superlinear number of rounds.

Using the observation that our solutions to the work problem yield solutions to Byzantine agreement, we can now return to the assumption that initially there is common knowledge about the work to be performed. Specifically, if even one process knows about this work, then it can act as a general, run Byzantine agreement on the pool of work using one of the three algorithms, and then the actual work is performed by running the same algorithm a second time on the real work. If n, the amount of actual work, is $\Omega(t)$, then the overall cost at most doubles when the work is not initially common knowledge.

1.1. Related work. The idea of doing work in the presence of failures, in a different context, has appeared elsewhere. First, Bridgland and Watro [5] considered a system of t asynchronous processes that together must perform n independent units of work. The processes may fail by crashing and each process can perform at most one unit of work. They provide tight bounds on the number of crash failures that can be tolerated by any solution to the problem.

Clearly, our problem assumes a very different model than that of [5]. Furthermore, they want a protocol that guarantees that the work will be performed in every execution of the protocol, while we want only a protocol that guarantees that the work

will be performed in executions in which at least one process survives. Consequently, their problem is not always solvable and their focus is on finding conditions under which it is solvable. Our problem is always solvable; our focus is on finding efficient solutions.

Another similar but not identical problem was considered by Kanellakis and Shvartsman. In a seminal paper [13] they consider the *Write-All* problem, in which n processes cooperate to set all n entries of an n-element array to the value 1. They provide an efficient solution that tolerates up to $n-1$ faults, and they show how to use it to derive robust versions of parallel algorithms for a large class of interesting problems. Their original paper was followed by a number of papers that consider the problem in other shared-memory models (see [1, 6, 14, 15, 16]).

The Write-All problem is, of course, a special case of the type of work we consider. Nevertheless, our framework differs from that of [13] in two important respects, so that their results do not apply to our problem (nor ours to theirs). First, they consider the shared-memory model while we consider the message-passing model. Using the shared-memory model simplifies things considerably for our problem. In this model, there is a straightforward algorithm (that uses shared memory to record what work has been done) with optimal effort $O(n+t)$ (where effort now counts both reading and writing into shared memory, as well as doing work), running in time $O(nt)$. While there are well-known emulators that can translate algorithms from the shared-memory model to the message-passing model (see [2, 3]), these emulators are not applicable for our problem, because the number of failures they tolerate is less than a majority of the total number of processes, while our problem allows up to $t-1$ failures. Also, these transformations introduce a multiplicative overhead of message complexity that is polynomial in t, while one of our goals here is to minimize this term.[1] Second, our complexity measure is inherently different from that of [13]. Kanellakis and Shvartsman's complexity measure is the sum, over the rounds during which the algorithm is running, of the number of processes that are not faulty during each round. They call their measure the *available processor steps*. This measure essentially "charges" for a nonfaulty process at round r whether or not it is actually doing any work (say, reading or writing a cell in shared memory). Our approach is generally *not* to charge a process in round r if it is not expending any effort (sending a message or performing a unit of work) at that round, since it is free at that round to be working on some other task.[2] Of course, the appropriateness of charging or not charging for process idle time will depend very much on the details of the system and the tasks being performed.

Our results have been extended by De Prisco, Mayer, and Yung [8] and by Galil, Mayer, and Yung [11]. De Prisco, Mayer, and Yung [8] consider the problem introduced here, but their goal is to optimize the available processor steps defined by [13], and then the number of messages. They present a message-efficient algorithm that achieves optimality in the available processor steps measure. They also show that when $t \approx n$, any algorithm for performing work in the message-passing model requires at least n^2 available processor steps. This lower bound can be avoided in shared-memory models that allow concurrent writes; for example, an $O(n \log^2 n)$ so-

[1] In fact, these emulators are designed for asynchronous systems, and it may be possible to improve their resilience for our synchronous model. Nevertheless, they seem to have an inherent multiplicative overhead in message complexity that is at least linear in t.

[2] Inactive processes in our algorithms may need to both receive messages and count the number of rounds that have passed, say, from the time they received their last message. We assume that processes can do this while carrying out other tasks.

lution is presented in [13]. Galil, Mayer, and Yung [11] employ the results of [8] to obtain a Byzantine agreement algorithm for the crash fault model that requires only a linear number of messages. Roughly speaking, the processors are organized into a tree. The children of the root attempt to solve the problem recursively; the group membership protocol of [8] (called a *checkpoint protocol*, not to be confused with the checkpoints as defined in our paper) is used to attempt to determine which of the children have failed to complete the recursive step, and the computation is reorganized accordingly.

The Galil, Mayer, and Yung protocol compares with that obtained by using our Protocol $\mathcal{B}$ as follows: [11] requires $O(n)$ messages, while ours requires $O(n\sqrt{n})$; [11] requires $O(n^{1+1/\epsilon})$ rounds of communication while ours requires $O(n)$; finally, [11] requires messages of length $\Omega(n + \log_2 |V|)$, where V is the set of possible agreement values. This appears to be because the protocol requires knowledge of which processors are alive and which processors occupy which parts of the tree. In contrast, our messages are of length $O(\log n + \log_2 |V|)$.

2. A protocol with effort $O(n+t^{3/2})$. Our goal in this section is to present a protocol with effort $O(n+t\sqrt{t})$ and running time $O(n+t)$. We begin with a protocol that is somewhat simpler to present and analyze, with effort $O(n+t\sqrt{t})$ and running time $O(nt+t^2)$. This protocol has the additional property of working with minimal change in an asynchronous environment with failure detection.

The main idea of the protocol is to use checkpointing in order to avoid redoing too much work if a process fails. The most naïve approach to checkpointing does not work. To understand why, suppose a process does a checkpoint after every n/k units of work. This means that up to n/k units of work are lost when a process fails. Since up to t processes may fail, this means that nt/k units of work can be lost (and thus must be repeated), which suggests we should take $k \geq t$ if we want to do no more than $O(n)$ units of work altogether. However, since each checkpoint involves t messages, this means that roughly tk messages will be sent. Thus we must have $k \leq \sqrt{t}$ if we are to use fewer than $t\sqrt{t}$ messages. Roughly speaking, this argument shows that doing checkpoints too infrequently means that there might be a great deal of wasted work, while doing them too often means that there will be a great deal of message overhead. Our protocol avoids these problems by doing *full* checkpoints to all the processes relatively infrequently—after $n/\sqrt{t}$ units of work—but doing partial checkpoints to only $\sqrt{t}$ processes after every n/t units of work. This turns out to be just the right compromise.

2.1. Description of the algorithm. For ease of exposition, we assume that t is a perfect square and that n is divisible by t (so that, in particular, $n > t$). We leave to the reader the easy modifications of the protocol when these assumptions do not hold. We assume that the processes are numbered 0 through $t-1$ and that the units of work are numbered 1 through n. We divide the processes into $\sqrt{t}$ groups of size $\sqrt{t}$ each and use the notation g_i to denote process i's group. (Note $g_i = \lceil (i+1)/\sqrt{t} \rceil$.) We divide the work into $\sqrt{t}$ *chunks*, each of size $n/\sqrt{t}$, and subdivide the chunks into $\sqrt{t}$ *subchunks* of size n/t.

The protocol guarantees that at each round, at most one process is *active*. The active process is the only process performing work. If process i is active, then it knows that processes 0 to $i-1$ have crashed or terminated. Initially, process 0 is active. The algorithm for process 0 is straightforward: process 0 starts out doing the work, a subchunk at a time. After completing a subchunk c, it does a checkpoint to the remaining processes in its group g_0 (processes 1 to $\sqrt{t}-1$); that is, it informs its group

that the subchunk of work has been completed by broadcasting to the processes in its group a message of the form (c). (If process 0 crashes in the middle of a broadcast, we make no assumption about which processes receive the message.) We call this a *partial checkpoint*, since the checkpointing is only to the processes in g_0. (Code for this for module and the whole protocol may be found in Figure 1.) After completing a whole chunk of work—that is, after completing a subchunk c which is a multiple of $\sqrt{t}$—process 0 informs *all* the processes that subchunk c has been completed, but it informs them one group at a time. After informing a whole group, it checkpoints the fact that a group has been informed to its own group (i.e., group 1). Formally, after completing a subchunk c that is a multiple of $\sqrt{t}$, process 0 does a partial checkpoint to its own group, and then for each group $2, \ldots, \sqrt{t}$, process 0 broadcasts to the processes in group g a message of the form (c, g) and then broadcasts to all the processes in its own group a message of the form (c, g). We call this a *full checkpoint*. Note that in a full checkpoint there is really a double checkpointing process: we checkpoint both the fact that work has been completed, and (to the processes in g_0) the fact that all processes have been informed that the work has been completed. Process 0 terminates after sending the message $(t, \sqrt{t})$ to process $t - 1$, indicating to the last process that the last chunk of work has been completed (unless it crashes before that round).

If process 0 crashes, we want process 1 to become active; if process 1 crashes, we want process 2 to become active, and so on. More generally, if process j discovers that the first $j - 1$ processes have crashed, then it becomes active. Once process j becomes active, it continues with essentially the same algorithm as process 0, except that it does not repeat the work it knows has already been done. We must ensure that the takeover proceeds in a "smooth" manner, so that there is at most one active process at a time.

Process j's algorithm is as follows. If j does not know that all the work has already been performed and sufficiently long time has passed from the beginning of the execution, then j becomes active. "Sufficiently long" means long enough to ensure that processes $0, \ldots, j-1$ have crashed or terminated. As we show below, we can take "sufficiently long" to be defined by the function $DD(j) = j(n+3t)$. ("DD" stands for deadline. We remark that this is not an optimal choice for the deadline; we return to this issue later.) Thus if the round number r is less than $DD(j)$, then j does nothing. Otherwise, if j does not know that the work is completed, it takes over as the active process at round $DD(j)$.

When j takes over as the active process, it essentially follows process 0's algorithm. Suppose the last message j received was of the form (c, g), and this message was received from a process k. By the syntax of the message we have that c is a multiple of $\sqrt{t}$ and that k was performing a full checkpoint when it sent the message. If $k \notin g_j$ then $g = g_j$, since this is the only kind of full checkpoint message that k sends to processes outside its group. Thus j informs the rest of its own group that subchunk c was performed, which it does with a Partialcheckpoint(c), and then proceeds with the full checkpoint of c, beginning with group $g_i + 1 = g + 1$.

If $k \in g_j$ then $g > g_j$; the meaning of (c, g) in this case is that k has told group g that subchunk c has been completed and is telling its own group g_k $(= g_j)$ about this fact. In this case j first ensures that its own group knows that group g has been informed about subchunk c, which it does by broadcasting (c, g) to the remainder of its group, and then proceeds with the full checkpoint beginning with group $g + 1$.

If the last message received was of the form (c) then this message was part of a partial checkpoint to g_j. In this case process j completes the partial checkpoint.

Main protocol

1. **if** round number $= DD(j)$ *and* not received (t) or (t, g_j)
2. **then** DoWork.

DoWork

1. **if** the last message received was from k and had the form (c, g)
2. **then** **if** $k \notin g_j$
3. **then** Partialcheckpoint(c); {see code below}
4. **else** Broadcast (c, g) to processes $j+1, \ldots, g_j\sqrt{t}-1$;
5. Fullcheckpoint$(c, g+1)$; {complete a full checkpoint; see code below}
6. **else** let (c) be the last message received;
7. Partialcheckpoint(c);
8. **if** c is a multiple of $\sqrt{t}$
9. **then** Fullcheckpoint(c, g_j+1).
10. **for** $s = c+1$ to t **do** {proceed with performing work}
11. Perform subchunk s;
12. Partialcheckpoint(s);
13. **if** s is a multiple of $\sqrt{t}$;
14. **then** Fullcheckpoint(s, g_j+1)

Partialcheckpoint(c)

1. Inform the remainder of group g_j that subchunk c has been performed by broadcasting (c) to processes $j+1, \ldots, g_j\sqrt{t}-1$

Fullcheckpoint(c, l)

1. **for** $g = l$ to $\sqrt{t}$ **do**
2. Inform group g that subchunk c has been performed by broadcasting (c, g) to group g;
3. Inform the remainder of group g_j that group g has been informed about subchunk c by broadcasting (c, g) to processes $j+1, \ldots, g_j\sqrt{t}-1$

FIG. 1. *Protocol $\mathcal{A}$: code for process j.*

In all three cases, j proceeds with work beginning with subchunk $c+1$ (if such a subchunk exists).

Unless it has already crashed, process j terminates before becoming active if it receives (t) (as part of a partial checkpoint) or (t, g_j) (as part of a full checkpoint). Otherwise, after becoming active at $DD(j)$, it terminates as follows. If $g_j = \sqrt{t}$ then j terminates after broadcasting (t) to the remainder of g_j. If $g_j < \sqrt{t}$ then j terminates after completing a call of the form Fullcheckpoint(t, g_j+1). This completes the description of our first protocol. We call this Protocol $\mathcal{A}$; the code appears in Figure 1.

Notice that we can easily modify this algorithm to run in a completely asynchronous system equipped with an appropriate *failure detection mechanism* [7]: if a process fails, then the failure detection mechanism must eventually inform all the processes that have not failed of this fact; moreover, the mechanism must be sound, in that it never says that a nonfaulty process has failed. The modification is trivial: rather than waiting until round $DD(j)$ before becoming active, process j waits until it has been informed that processes $1, \ldots, j-1$ crashed or terminated.

2.2. Analysis and proof of correctness. We now give a correctness proof for Protocol $\mathcal{A}$. We say a process is *retired* if it has either crashed or terminated.

LEMMA 2.1. *A process performs at most n units of work, sends at most $3t\sqrt{t}$ messages, and runs for less than $n+3t$ rounds from the time it becomes active to the time it retires.*

Proof. It is easy to see that from the time process i becomes active, it performs each unit of work at most once, partial checkpoints each subchunk at most once (and hence performs at most t partial checkpoints), and full checkpoints every chunk at most once (and hence performs at most $\sqrt{t}$ full checkpoints). Each partial checkpoint consists of a broadcast to process i's group and hence involves at most $\sqrt{t}$ messages and one round. Thus process i spends at most t rounds on partial checkpoints and sends at most $t\sqrt{t}$ messages when performing partial checkpoints. During a full checkpoint, process i broadcasts once to each group other than its own, and broadcasts at most $\sqrt{t}$ times to its own group. Each broadcast involves at most $\sqrt{t}$ messages and one round, and there are $\sqrt{t}$ groups. Thus process i sends less than $2t\sqrt{t}$ messages when performing full checkpoints and takes less than $2t$ rounds doing so. The required bounds immediately follow. □

Recall that $DD(j) = j(n+3t)$. The following lemma is now immediate from the definition of DD.

LEMMA 2.2. *Assume process j becomes active at round r of an execution $e_{\mathcal{A}}$ of Protocol $\mathcal{A}$. Then all processes $< j$ have retired before round r.*

It is sometimes convenient to view a group g_i as a whole. Therefore we say that a *group is active* in the period starting when some process in this group becomes active and ending when the last process of this group retires. Notice that Lemma 2.2 ensures that when g_i becomes active, all processes in smaller groups have retired.

THEOREM 2.3. *In every execution of Protocol $\mathcal{A}$,*

(a) *at most $3n$ units of work are performed in total by the processes;*

(b) *at most $9t\sqrt{t}$ messages are sent;*

(c) *by round $nt + 3t^2$, all processes have retired.*

Proof. Part (c) is immediate from Lemma 2.1 and the definition of DD.

We prove parts (a) and (b) simultaneously. To do so, we need a careful way of counting the total number of messages sent and the total amount of work done. A given unit of work may be performed a number of times. If it is performed more than once, say by processes $i_1, \ldots, i_k$, we say that i_2 *redoes* that unit of work of i_1, i_3 redoes the work of i_2, etc. It is important to note that i_3 does not redo the work of i_1 in this case, only that of i_2. Similarly, we can talk about a message sent during a partial checkpoint of a subchunk or a full checkpoint of a chunk done by i_1 as being *resent* by i_2. In particular, a message m sent by i_1 as part of a broadcast is resent by i_2 if i_2 sends exactly the same message as part of a broadcast (not necessarily to the same set of recipients). For example, if i_1 sends (c) to the remainder of g_{i_1} as part of a partial checkpoint, and later i_2 sends (c) to the remainder of g_{i_2}, then, whether or not $g_{i_1} = g_{i_2}$, the messages in the second broadcast are considered to be resendings.

Since the completion of a chunk is followed by a full checkpoint, it is not hard to show that when a new group becomes active it will redo at most one chunk of work that was already done by previous active groups. It will also redo at most one full checkpoint that was done already on the previous chunk and at most $\sqrt{t}$ partial checkpoints (one for each subchunk of work redone). In all, it is easy to see that at most $n/\sqrt{t}$ units of work done by previous groups are redone when a new group becomes active and at most $3t$ messages are resent. Similarly, since the completion of a subchunk is followed by a partial checkpoint, it is not hard to show that when a new process, say i, in a group that is already active becomes active, and the last message it received was of the form (c) (i.e., a partial checkpoint of subchunk c), it will redo at most one subchunk that was already done by previous active process (namely, $c+1$) and may possibly resend the messages in two partial checkpoints: the one sent after subchunk c and the one sent after subchunk $c+1$ (if the previous process crashed during the checkpointing of $c+1$ without i receiving the message). If the last message that i received was (c, g) for $g > g_i$ (that is, the checkpointing of a checkpoint in the middle of a full checkpoint), then similar arguments show that it may resend $3\sqrt{t}$ messages: the checkpoint of (c, g) to its own group, the checkpoint $(c, g+1)$ to group $g+1$, and the checkpointing of $(c, g+1)$ to its own group. Thus the amount of work done by an active group that is redone when a new process in that group becomes active is at most n/t, and the number of messages resent is at most $3\sqrt{t}$.

The maximum amount of unnecessary work done is (number of groups) $\times$ (amount of work redone when a new group becomes active) + (number of processes) $\times$ (amount of work redone when a new process in an already active group becomes active) $\leq \sqrt{t}(n/\sqrt{t}) + t(n/t) = 2n$. Similarly, the maximum number of unnecessary messages that may be sent is no more than (number of groups) $\times$ (number of messages resent when a new group becomes active) + (number of processes) $\times$ (number of messages resent when a new process in an already active group becomes active) $\leq \sqrt{t}(3t) + t3\sqrt{t} = 6t\sqrt{t}$. Clearly, n units of work must be done; by Lemma 2.1, at most $3t\sqrt{t}$ messages are necessary. Thus no more than $3n$ units of work will be done altogether, and no more than $9t\sqrt{t}$ messages will be sent altogether. □

2.3. Improving the time complexity. As we have observed, the round complexity of Protocol $\mathcal{A}$ is $nt + 3t^2$. We now discuss how the protocol can be modified to give a protocol that has round complexity $O(n+t)$, while not significantly changing the amount of work done or the number of messages sent.

Certainly one obvious hope for improvement is to use a better function than DD for computing when process i should become active. While some improvement is possible by doing this, we can get a round complexity of no better than $O(n\sqrt{t})$ if this is all we do, which is still more than we want. Intuitively, the problem is that if process j gets a message of the form (c, g), then it is possible, as far as j is concerned, that some other process $i < j$ may have received a message of the form $(c+\sqrt{t}, h)$. (Observe that this situation is possible even if $g_i = g_j$ because if the sender of the message $(c+\sqrt{t}, h)$ crashes at the round when it broadcasts this message to g_i, this message may reach an arbitrary subset of the processes in g_i.) Process j cannot become active before it is sure that i has retired. To compute how long it must wait before becoming active, it thus needs to compute how long i would wait before becoming active, given that i got a message of the form $(c+\sqrt{t}, h)$. On the other hand, if i did get such a message, then as far as i is concerned, some process $i' < i$ may have received a message of the form $(c+2\sqrt{t}, h')$. Notice that, in this case, process j knows perfectly well that no process received a message of the form $(c+2\sqrt{t}, h')$; the

problem is that i does not know this and must take into account this possibility when it computes how long to wait before becoming active. Carrying out a computation based on these arguments gives an algorithm which runs in $O(n\sqrt{t})$ rounds.

On closer inspection, it turns out that the situation described above really causes difficulties only when all processes involved (in the example above, this would be the processes j, i, and i') are in the same group. Thus in our modified algorithm, called Protocol $\mathcal{B}$, process j computes the time to become active as follows: suppose that the last message received by process j before round r was received from process i in round r'. Process j then computes a function $DD_{\mathcal{B}}(j,i)$ with the property that if $r = r' + DD_{\mathcal{B}}(j,i)$, then process j knows at round r that all processes in groups $g' < g_j$ must have retired. Moreover, if $g_i = g_j$, then j knows at round r that all processes $\leq i$ must have retired. Process j then polls all the lower-numbered processes in its own group not known to it as retired, one by one, to see if they are alive; if not, then j becomes active. If any of them is alive, then the lowest-numbered one that is alive becomes active upon receipt of j's message. Once a process becomes active, it proceeds just as in Protocol $\mathcal{A}$. This technique turns out to save a great deal of time, while costing relatively little in the way of messages.

In particular, in Protocol $\mathcal{B}$, process 0 follows the same algorithm as in Protocol $\mathcal{A}$. Process j's algorithm is as follows. Here j receives messages either of the form (c), (c,g) or of the form **go ahead**. We call the first two types of messages *ordinary*, to distinguish them from the **go ahead** messages. Suppose that the last ordinary message received by process j before round r is of the form (c) or (c,g), and this message was received from process i at round r'. To avoid dealing separately with the special case in which j does not receive any message before it becomes active, we use the convention that process 0 becomes active in round 0 (just before the execution begins) and every process receives from it an ordinary message $(0,g)$ at that round. (These fictitious messages are used only in the analysis and hence will not be taken into account when computing the message complexity of the protocol. Also, if in the actual execution process 0 crashes before ever becoming active then we say that it crashes just after it finishes broadcasting these fictitious messages.) There are now two ways for j to become active at round r. The first is if j receives a **go ahead** message at round r and $c < t$. In this case j becomes active, proceeding just as in Protocol $\mathcal{A}$ when it became active at round $DD(j)$. Alternatively, if j does not receive a message for a sufficiently long time, j becomes active. Intuitively, sufficiently long will ensure that all processes smaller than j have already retired.

To analyze this more formally, we need some definitions. Let PTO be $n/t + 2$. ("PTO" stands for process time out.) $PTO - 1$ is an upper bound on the number of rounds that can pass before process j in group $g_j = g_i$ hears from process i if i is active. Let $\bar{\jmath}$ denote j mod $\sqrt{t}$. Let $GTO(i)$ be $n/\sqrt{t} + 3\sqrt{t} + (\sqrt{t} - \bar{\imath} - 1)PTO + 1$. ("$GTO$" stands for group time out.) $GTO(i) - 1$ is an upper bound on the number of rounds that can pass before process j in group $g_j > g_i$ hears from a process k in g_i with $\bar{\imath} \leq \bar{k} \leq \sqrt{t} - 1$ if any of these processes is active. Next we define a new deadline function as follows:

$$DD_{\mathcal{B}}(j,i) = \begin{cases} GTO(i) + (g_j - g_i - 1)GTO(0) & \text{if } g_j \neq g_i, \\ PTO & \text{otherwise.} \end{cases}$$

We now define "sufficiently long" in terms of $DD_{\mathcal{B}}$ rather than DD. Again taking r to be the current round, r' to be the last round before r in which j received a message, and i to be the process sending that message, j proceeds as follows: if $r < r' + DD_{\mathcal{B}}(j,i)$, then j does nothing. If $c < t$ and $r = r' + DD_B(j,i)$, j becomes *preactive*. First

consider the case where $g_i \neq g_j$. Informally, at this point, j knows that all processes from groups $< g_j$ have failed. In this case, it sends a **go ahead** message to each lower-numbered process in its group, starting with the first process in g_j up to the $(\bar{\jmath}-1)$st process in g_j, and waiting $PTO-1$ rounds between messages to see if it receives a message. (Observe that if the recipient of the **go ahead** message is alive, the sender receives a message from it within one round after the **go ahead** mesage was sent; however, for technical reasons the sender of the **go ahead** messages waits $PTO-1$ rounds between two successive **go ahead** messages.) If $g_i = g_j$, process j proceeds similarly to the case where $g_i \neq g_j$ except that when sending **go ahead** messages it starts with the $(\bar{\imath}+1)$st process in g_j. That is, it sends a **go ahead** message to each lower-numbered process in its group, starting with the $(\bar{\imath}+1)$st process in g_j up to the $(\bar{\jmath}-1)$st process in g_j, and waiting $PTO-1$ rounds between messages to see if it receives a message. If j does not receive any response to its **go ahead** messages by round $r' + DD_{\mathcal{B}}(j,i) + \bar{\jmath}PTO - 1$ if $g_i \neq g_j$, or by round $r' + (\bar{\jmath}-\bar{\imath})PTO - 1$ if $g_i = g_j$, then it becomes active at round $r' + DD_{\mathcal{B}}(j,i) + \bar{\jmath}PTO$ (respectively, $r' + (\bar{\jmath}-\bar{\imath})PTO$), proceeding just as it did in Protocol $\mathcal{A}$ at round $DD(j)$. If it does get a message, then j becomes passive again.

Note that the construction of the algorithm guarantees that if $r = r' + DD_{\mathcal{B}}(j,i) + \bar{\jmath}PTO$, the last ordinary message that j receives in execution $e_{\mathcal{B}}$ prior to round r was sent by i, and $g_i \neq g_j$, then j will become active in $e_{\mathcal{B}}$ at or before round r. (It may become active earlier if it receives a **go ahead** message.) Similarly, if $r = r' + (\bar{\jmath}-\bar{\imath})PTO$, the last ordinary message that j receives in execution $e_{\mathcal{B}}$ prior to round r was sent by i, and $g_i = g_j$, then j will become active in $e_{\mathcal{B}}$ at or before round r. Define

$$TT(j,i) = \begin{cases} GTO(i) + (g_j - g_i - 1)GTO(0) + \bar{\jmath}PTO & \text{if } g_j \neq g_i, \\ (\bar{\jmath}-\bar{\imath})PTO & \text{otherwise.} \end{cases}$$

("TT" stands for transition time.) Our observations above show that if $r = r' + TT(j,i)$ and the last ordinary message that j receives in execution $e_{\mathcal{B}}$ prior to round r was sent by i, then j will become active in $e_{\mathcal{B}}$ at or before round r.

Unless it has already crashed, process j terminates before becoming active if it receives (t) (as part of a partial checkpoint) or (t, g_j) (as part of a full checkpoint). Otherwise, after becoming active it terminates as follows. If $g_j = \sqrt{t}$ then j terminates after broadcasting (t) to the remainder of g_j. If $g_j < \sqrt{t}$ then j terminates after completing a call of the form Fullcheckpoint$(t, g_j + 1)$. This completes the description of Protocol $\mathcal{B}$. The code for Protocol $\mathcal{B}$ appears in Figure 2; it uses the code for the DoWork procedure in Figure 1.

2.4. Proof of correctness of Protocol $\mathcal{B}$. In this section we show that the round complexity of Protocol $\mathcal{B}$ is $O(n+t)$, and that neither the amount of work done nor the number of messages sent in Protocol $\mathcal{B}$ is significantly larger than that in Protocol $\mathcal{A}$.

Suppose for a moment that in every execution of Protocol $\mathcal{B}$ a process becomes active only after all lower-numbered processes have retired. Since when a process becomes active in an execution of Protocol $\mathcal{B}$ it performs essentially the same steps as when it becomes active when it follows Protocol $\mathcal{A}$, a similar proof to the one of Theorem 2.3 will show that the amount of work performed in any execution of Protocol $\mathcal{B}$ is no more than $3n$ units (which is roughly the maximum amount of work performed in any execution of Protocol $\mathcal{A}$), and the number of ordinary messages sent is no more than $9t\sqrt{t}$ (which is roughly the maximum number of messages sent in any execution

Main protocol

1. **if** just received a **go ahead** message
2. **then** DoWork; {see Figure 1 for details}
3. **else** **if** round number $= r' + DD_{\mathcal{B}}(j, i)$
   and last message received was from process i at round r'
4. **then** PreactivePhase(i, r')

PreactivePhase(i, r')

1. **if** $g_i \neq g_j$
2. **then** $i' := (g_j - 1)\sqrt{t}$;
3. **else** $i' := i + 1$;

4. $r :=$ *current round*;
5. **while** not received a message *and* $i' < j$ **do**
6. **if** *current round* $- r \equiv 0 \pmod{PTO}$
7. **then** send **go ahead** message to process i';
8. $i' := i' + 1$;
9. **if** just received a **go ahead** message or $i' \geq j$
10. **then** DoWork

FIG. 2. *Protocol $\mathcal{B}$: code for process j.*

of Protocol $\mathcal{A}$). Since the number of **go ahead** messages sent in any execution of Protocol $\mathcal{B}$ is at most $t\sqrt{t}$ (each process sends at most one **go ahead** message to every other process in its group), it follows immediately that the total amount of effort performed by Protocol $\mathcal{B}$ is not significantly larger than the one performed by Protocol $\mathcal{A}$. Therefore, the main property we need to prove is that in every execution of Protocol $\mathcal{B}$, a process becomes active only after all lower-numbered processes have retired.

Our analysis uses what we call *activation chains*. The round r *activation chain* of process i, denoted $ac(i, r)$, is the sequence of processes $\langle i_m, \ldots, i_0 \rangle$ such that $i_0 = i$ and for all j, if i_j received an ordinary message prior to round r, then i_{j+1} is the sender of the last message received by i_j. (As we show below, it cannot be the case that i_j receives ordinary messages from two distinct processes in the same round. Since we have not yet proved this, for now, if i_j received ordinary messages from more than one process in the last round in which it received an ordinary message, we take i_{j+1} to be the lowest-numbered process among them.) Notice that our convention that process 0 sent a round 0 message guarantees that $i_m = 0$. In addition, note that the processes in the activation chain appear in increasing order since a process sends messages only to higher-numbered processes.

It is sometimes convenient to view i's activation chain as a whole and to reason about the effort performed by the chain. We say that process k in $ac(i, r)$ is the *current process* from the round it becomes active up to (but not including) the round at which its successor in the chain becomes active. Note that a process in $ac(i, r)$ is current when it first becomes active. Now let k, l be processes in $ac(i, r)$ such that l immediately succeeds k, and assume the last ordinary message l receives from k

before l becomes active is sent at round r'. Clearly, any operation (sending messages or performing a unit of work) done by k after round r' is not known by l and hence may be repeated by the chain (that is, may be repeated by some process when it is the current process in the chain). On the other hand, any operation done by k before round r' will be known by the processes succeeding it in the chain by the time they become active and hence will not be repeated by the chain. The operation done by k in round r', which is a broadcast to g_l, will be repeated by l in the first round in which l becomes active. We say that an operation performed by a process k in the chain is *useful* if it is performed before the round in which the process immediately succeeding k in the chain heard from k for the last time before becoming active (if the process is i, then there is no later process in the chain, and hence all operations performed by i are useful). When we refer to an *operation performed by a chain* $ac(l, r)$, we mean a useful operation performed by some process in that chain. We say that a round is *useful* for the chain $ac(l, r)$ if the chain performed a useful operation in that round; otherwise we say that the round is *useless*.

The discussion above shows that the operations performed by a chain proceed in a similar order to the operations performed by a single active process. More precisely, if we consider only useful operations, the processes in the chain perform work units one by one in the natural order and without repetition; each time a subchunk c is completed by the chain, the group of the process that completes this subchunk is informed about this fact exactly once, and if the completed subchunk is a multiple of $\sqrt{t}$, then in addition all groups whose numbers are higher than the group of the process that completed this subchunk are informed that the subchunk is completed one by one in the natural order and without repetition (that is, each such group g receives a message (c, g) exactly once); moreover, each time such a group g is informed, this fact is checkpointed to the group of the informer exactly once. Assume process i is active at some round r with $r \geq r2 \geq r1 \geq 1$. Then we denote by $T_{r1}^{r2}(i)$ the number of useful rounds for the chain $ac(i, r)$ in interval $[r1, r2]$ (that is, in the period from round $r1$ to round $r2$ (inclusive)).

LEMMA 2.4. *Let l be active at some round r with $r \geq r2 \geq r1 \geq 1$. Then*

(a) $T_{r1}^{r2}(l) \leq n + 3t$;

(b) *if $T_{r1}^{r2}(l) \geq n/\sqrt{t} + 3\sqrt{t}$, then each process $\geq l$ must have received a message from (a process in) $ac(l, r)$ at some round r' such that $r1 \leq r' \leq r2$.*

Proof. Part (a) follows from the fact that in each useful round, the chain does one of the following: it performs work, it checkpoints to some group g the fact that a subchunk c was performed, or it checkpoints the fact that group g was informed that chunk c was performed. The discussion above shows that no unit of work is repeated and hence there are at most n useful rounds in which the chain performs work. Similarly, each subchunk is partially checkpointed at most once and hence there are at most t useful rounds in which the chain performs partial checkpoints of subchunks. Also, the completion of a chunk is checkpointed to each group at most once, yielding at most t useful rounds in which such subchunks are checkpointed. Finally, the fact that group g was informed about chunk c is checkpointed at most once, yielding at most t additional useful rounds. Summing the above the claim follows.

Part (b) follows because, as reasoned above, the useful operations done by the chain follow the same order as if they are done by a single active process. Hence within $n/\sqrt{t} + 3\sqrt{t}$ rounds the chain must complete a chunk and a full checkpoint. ∎

Now, as we mentioned above, at the core of our proof of correctness is the fact that when a process becomes active, all lower numbered processes have already retired. To prove this, we first prove a lower bound on the number of useful rounds for a given activation chain in a given period. Using this bound, we can show that if some process i receives its last ordinary message before becoming active at round $r1$, i becomes active at round $r2$, and some process $l < i$ has not retired by $r2$, then process i would have received an ordinary message from some process in $ac(l, r2)$ between rounds $r1$ and $r2$, contradicting our choice of $r1$.

We now proceed with the formal proofs. We start with a technical lemma.

LEMMA 2.5. *Let $l > j > k$. Then*

(a) $TT(j,k) + TT(l,j) = TT(l,k)$;

(b) *if $g_j < g_l$, then $TT(j,k) + DD_{\mathcal{B}}(l,j) = DD_{\mathcal{B}}(l,k)$.*

Proof. The proof is straightforward. We start with part (a). In the calculations below, we use "$(g_i = g_j)$" to denote the value 1 if $g_i = g_j$ and 0 otherwise. Similarly, "$(g_i \neq g_j)$" denotes 1 if $g_i \neq g_j$, and 0 otherwise. Recall that $\bar{\jmath}$ denotes j mod $\sqrt{t}$.

$$\begin{aligned} TT(j,k) + TT(l,j) = {} & [GTO(k) + (g_j - g_k - 1)GTO(0) + \bar{\jmath}PTO](g_j \neq g_k) \\ & + [(\bar{\jmath} - \bar{k})PTO](g_j = g_k) \\ & + [GTO(j) + (g_l - g_j - 1)GTO(0) + \bar{l}PTO](g_l \neq g_j) \\ & + [(\bar{l} - \bar{\jmath})PTO](g_l = g_j). \end{aligned}$$

If $g_j = g_k$, then

$$\begin{aligned} TT(j,k) + TT(l,j) &= (\bar{\jmath} - \bar{k})PTO \\ &\quad + [GTO(j) + (g_l - g_k - 1)GTO(0) + \bar{l}PTO](g_l \neq g_k) \\ &\quad + [(\bar{l} - \bar{\jmath})PTO](g_l = g_k) \\ &= [GTO(k) + (g_l - g_k - 1)GTO(0) + \bar{l}PTO](g_l \neq g_k) \\ &\quad + [(\bar{l} - \bar{k})PTO](g_l = g_k) \\ &= TT(l,k), \end{aligned}$$

and part (a) follows. (In the first equality we replaced g_j by g_k since in this case they are identical, and the second equality follows because $GTO(j) + (\bar{\jmath} - \bar{k})PTO = GTO(k)$.)

If $g_j \neq g_k$, then

$$\begin{aligned} TT(j,k) + TT(l,j) &= [GTO(k) + (g_j - g_k - 1)GTO(0) + \bar{\jmath}PTO] \\ &\quad + [GTO(j) + (g_l - g_j - 1)GTO(0) + \bar{l}PTO](g_l \neq g_j) \\ &\quad + [(\bar{l} - \bar{\jmath})PTO](g_l = g_j) \\ &= [GTO(k) + (g_l - g_k - 1)GTO(0) + \bar{l}PTO](g_l \neq g_j) \\ &\quad + [GTO(k) + (g_j - g_k - 1)GTO(0) + \bar{l}PTO](g_l = g_j) \\ &= [GTO(k) + (g_l - g_k - 1)GTO(0) + \bar{l}PTO] \\ &= TT(l,k), \end{aligned}$$

and again part (a) follows. (The second equality follows by a case analysis on whether or not $g_l = g_j$, using the fact that $GTO(j) + \bar{\jmath}PTO = GTO(0)$ and the fourth equality follows since $g_j \neq g_k$ and $l > j > k$ implies $g_l \neq g_k$.)

The proof of part (b) is similar. Observe that here by assumption, $g_l \neq g_j$ and hence also $g_l \neq g_k$. If $g_j = g_k$, then

$$\begin{aligned} TT(j,k) + DD_{\mathcal{B}}(l,j) &= (\bar{\jmath} - \bar{k})PTO + [GTO(j) + (g_l - g_k - 1)GTO(0)] \\ &= [GTO(k) + (g_l - g_k - 1)GTO(0)] \\ &= DD_{\mathcal{B}}(l,k), \end{aligned}$$

and part (b) follows.

If $g_j \neq g_k$, then

$$\begin{aligned} TT(j,k) + DD_{\mathcal{B}}(l,j) &= [GTO(k) + (g_j - g_k - 1)GTO(0) + \bar{\jmath}PTO] \\ &\quad + [GTO(j) + (g_l - g_j - 1)GTO(0)] \\ &= [GTO(k) + (g_l - g_k - 1)GTO(0)] \\ &= DD_{\mathcal{B}}(l,k), \end{aligned}$$

and part (b) follows. □

The next lemma establishes a lower bound on the number of useful rounds for an activation chain in a given interval.

LEMMA 2.6. *Assume l is active at some round r such that $r \geq r2 \geq r1 \geq 1$. Assume $p \geq k$ is the current process in $ac(l,r)$ at some round $\leq r1$. Then $T_{r1}^{r2}(l) \geq r2 - r1 - TT(l,k) + 1$.*

Proof. We first show that if j is in $ac(l,r)$ and becomes active at round r' with $r1 \leq r' \leq r2$, then there are at most $TT(j,k)$ useless rounds in $[r1, r'-1]$. We proceed by induction on r'. If $r' = r1$, the result is trivial. If $r' > r1$, then j is i's successor for some i in the activation chain and j received its last message from i at some round r''. (There is such an i and such a message since by convention process 0 sent an ordinary message to everybody just before the execution begins.) By definition, we have $r' \leq r'' + TT(j,i)$. If $i = k$, we are done, since no round in $[r1, r'' - 1]$ is useless, so there are at most $TT(j,k) = TT(j,i)$ useless rounds in $[r1, r' - 1]$. If $i > k$, then suppose i becomes active at r'''. By the inductive hypothesis, there are at most $TT(i,k)$ useless rounds in $[r1, r''' - 1]$. All the rounds in $[r''', r'' - 1]$ are useful. Thus there are at most $TT(j,i) + TT(i,k)$ useless rounds in $[r1, r' - 1]$. Since $TT(j,i) + TT(i,k) = TT(j,k)$ by Lemma 2.5, the inductive step follows.

Suppose that l becomes active at round $r3$. By the argument above, there are at most $TT(l,k)$ useless rounds in $[r1, r3 - 1]$. If $r3 > r2$, it immediately follows that there are at most $TT(l,k)$ useless rounds in $[r1, r2]$. On the other hand, if $r3 < r2$, since l is still active at $r > r2$, it follows that there are no useless rounds in $[r3, r2]$. Hence we again get that there are at most $TT(l,k)$ useless rounds in $[r1, r2]$. The lemma follows. □

The next lemma shows that in every execution of Protocol $\mathcal{B}$, by the time a process becomes active, all lower numbered processes have retired.

LEMMA 2.7. *In every execution of Protocol $\mathcal{B}$,*

(a) *before the round r in which process i becomes preactive, all processes in groups $< g_i$ have retired;*

(b) *before the round r in which process i becomes active, all processes $< i$ have retired.*

Proof. Fix an execution $e_{\mathcal{B}}$ of Protocol $\mathcal{B}$. The proof proceeds by induction on the round r. The base case of $r = 0$ holds trivially since only process 0 is active then. Assume the claim for $< r$, and we will show it for r. If $i = 0$, the claim holds trivially.

Thus we can assume $i > 0$. Suppose that the last ordinary message that i received before round r came from k, and was received at round $r1$. (Note that there must have been such an ordinary message, given our assumption that process 0 sent an ordinary message to all the processes before the execution begins.)

We first prove part (a). Assume, by way of contradiction, that some process l with $g_l < g_i$ does not retire by round r.

Since, by assumption, $r1$ was the latest round $< r$ at which i received an ordinary message, to complete the proof it is enough to show that if l does not retire before round r, i must have received an ordinary message at some round r'' with $r1 < r'' < r$. In fact, we plan to show that i must have received a message in the interval $(r1, r)$ from some process in $ac(l, r)$. To do this, we plan to use Lemmas 2.4 and 2.6. Notice that both of these lemmas require l to be active. In fact, we can assume without loss of generality that l is active at some round $r3 \geq r$ of $e_{\mathcal{B}}$, and that $ac(l, r) = ac(l, r3)$. If not, we can just consider the execution $e'_{\mathcal{B}}$ which is identical to $e_{\mathcal{B}}$ up to round r, after which all processes other than l crash. It is clear that eventually l becomes active in $e_{\mathcal{B}}$, with the same activation chain it has in round r. Moreover, if i receives an ordinary message in the interval $(r1, r)$ in $e'_{\mathcal{B}}$, then it must also receive the same message in $e_{\mathcal{B}}$, since the two executions agree up to round r.

Since k becomes active at some round prior to r, the inductive hypothesis on part (b) of the lemma implies that all processes $\leq k$ have retired by round $r1 < r$. Thus without loss of generality $l \geq k$. We consider two cases: (i) k is in $ac(l, r)$; (ii) k is not in $ac(l, r)$.

In case (i), since k is in l's activation chain and is active at round $r1$, by the inductive hypothesis, it must be the current process in $ac(l, r)$ at round $r1$. Applying Lemma 2.6 to $ac(l, r3) = ac(l, r)$, we get

$$T^{r-1}_{r1+1}(l) \geq (r-1) - (r1+1) - TT(l,k) + 1.$$

By definition, i becomes preactive in round $r = r1 + DD_{\mathcal{B}}(i, k)$, and hence $r - r1 = DD_{\mathcal{B}}(i, k)$. Substituting this into the above inequality we get

$$T^{r-1}_{r1+1}(l) \geq DD_{\mathcal{B}}(i,k) - TT(l,k) - 1.$$

Since $g_i > g_l$, Lemma 2.5 implies that $DD_{\mathcal{B}}(i, k) - TT(l, k) = DD_{\mathcal{B}}(i, l)$, and substituting this fact in the above inequality we get

$$\begin{aligned} T^{r-1}_{r1+1}(l) &\geq DD_{\mathcal{B}}(i,l) - 1 \\ &= GTO(l) + (g_i - g_l - 1)GTO(0) - 1 \\ &= (n/\sqrt{t} + 3\sqrt{t} + (\sqrt{t} - \bar{l} - 1)PTO + 1) + (g_i - g_l - 1)GTO(0) - 1 \\ &\geq n/\sqrt{t} + 3\sqrt{t}. \end{aligned}$$

Thus part (b) of Lemma 2.4 implies that i must have received an ordinary message at some round in the interval $(r1, r)$, contradicting the assumption that it does not, and the claim follows.

In case (ii), let k' be the greatest process $< k$ in l's activation chain, and let j be the smallest process $> k$ in l's activation chain. Suppose j gets its last message before becoming active from k' at round $r0$. (Note that this means that the last message received by j before becoming active came at $r0$.) Since the inductive hypothesis on part (b) implies that k' must retire before k becomes active, and since k must become active at least one round before it sent a message to i (since by assumption

$g_k \leq g_l < g_i$ and process k checkpoints to its own group before it sends a message to another group), we have $r0 < r1 - 1$. Furthermore, since the processes succeeding k' in l's chain are greater than k, the same inductive hypothesis implies that these processes can become active only after process k retires, and hence after round $r1$. Since by definition any message received by i after round $r0$ from l's chain must be sent by one of the processes succeeding k' in the chain, it follows that if i receives a message from l's chain after round $r0$, this message is sent after round $r1$.

To complete the proof we show that i must have received some message from l's chain at some round in the interval $(r0, r)$, and hence in the interval $(r1, r)$, contradicting the assumption that it does not. As argued above, to show this it is enough to show that $T^{r-1}_{r0+1}(l) \geq n/\sqrt{t} + 3\sqrt{t}$. Applying Lemma 2.6 to l's activation chain we get

$$T^{r-1}_{r0+1}(l) \geq r - 1 - (r0 + 1) - TT(l, k') + 1.$$

To bound $T^{r-1}_{r0+1}(l)$, we need to bound $r - r0$. To do this we will compute two terms: (a) $r - r1$ and (b) $r1 - r0$. The first term is equal to $DD_{\mathcal{B}}(i, k)$ as argued above. To compute the second term, we first show (1) $g_j > g_k$ and (2) $g_{k'} = g_k$.

For (1), clearly $g_j \geq g_k$, since $j > k$. If $g_j = g_k$, j must have received a message from k at round $r1 - 1$ before k sent a message to i (since by assumption $g_k \leq g_l < g_i$ and process k checkpoints to its own group just before it sends a message to another group). As we have observed, $r1 - 1 > r0$, so this contradicts the assumption that the last message received by j before becoming active came at $r0$. Thus $g_j > g_k$.

For (2), clearly $g_{k'} \leq g_k$. If $g_{k'} < g_k$, this means that j did not receive a message from a process in g_k before becoming active (because if it did, then by the inductive hypothesis on part (b) we have that this message arrives after k' retires and hence after round $r0$). But since $g_j \leq g_l < g_i$, and k sent a message to i at $r1$, some process in g_k must have sent a message to j before round $r1$, and hence before j becomes active. This gives us the desired contradiction.

To complete the proof of case (ii) we use the following claim.

CLAIM 2.1. *Every process k'' with $k' < k'' \leq k$ that becomes active does so no earlier than round $r0 + (\bar{k''} - \bar{k'})PTO - 1$.*

Proof. We proceed by induction. Assume $k' < k1 \leq k$ and the claim holds for all k'' with $k' < k'' < k1$. We prove it for $k1$.

We first show that the last ordinary message that any process $\geq k1$ in g_k receives from any process $k2$ with $k' \leq k2 < k1$ is sent no earlier than round $r0 + (\bar{k2} - \bar{k'})PTO - 1$. Observe that since k' is in g_k, so are $k1$ and $k2$. If $k2 = k'$, the claim trivially follows since k' must send a message to its own group at round $r0 - 1$ just before it sends a message to $g_j > g_k$. Otherwise, by the induction hypothesis we have that $k2$ became active no earlier than round $r0 + (\bar{k2} - \bar{k'})PTO - 1$, and the claim again follows.

Let $k2$ be the last process from which $k1$ receives an ordinary message. Observe that $k' \leq k2 < k1$. (Because, as reasoned above, $k1$ has received a message from k', and hence the message sent by $k2$ was sent at or after the time the message sent by k'. The inductive hypothesis on part (b) therefore implies that $k2 \geq k'$.) It follows from the claim above that the message from $k2$ was sent no earlier than round $r0 + (\bar{k2} - \bar{k'})PTO - 1$. In addition, the inductive hypothesis on part (b) implies that $k1$ becomes active only after $k2$ retires, and hence only after receiving its message.

Now assume that $k1$ does not receive a **go ahead** message. It then becomes preactive PTO rounds after it receives the last ordinary message from $k2$ and then $k1$

starts sending **go ahead** messages to lower-numbered processes in its group. Since, by assumption, $k1$ does not receive a message in response, it becomes active $TT(k1, k2) = (\bar{k1} - \bar{k2})PTO$ rounds after receiving this last message from $k2$, and hence no earlier than round $r0 + (\bar{k1} - \bar{k'})PTO - 1$.

Next assume $k1$ receives a **go ahead** message. Let $k3$ be the process sending this message. Let $k2$ be the last process from which $k3$ received an ordinary message before sending the **go ahead** message to $k1$. Since $k3$ sends a **go ahead** message to $k1$, it follows that $k2 < k1$. Just as above, we can show that $k2 \geq k'$, and hence that $k3$ received the ordinary message from $k2$ no earlier than round $\geq r0 + (\bar{k2} - \bar{k'})PTO - 1$. Clearly, $k3$ sends the **go ahead** message to $k1$ no earlier than $(\bar{k1} - \bar{k2})PTO$ rounds after it receives its ordinary message from $k2$, and the claim follows as above. This completes the proof of the inductive step. □

Now, to compute $r1 - r0$, observe that $r1$, the round in which k sends a message to i, is at least one round after k becomes active (because $g_i > g_k$ and k first broadcasts to its own group), and hence Claim 2.1 immediately implies that $r1 - r0 \geq (\bar{k} - \bar{k'})PTO$. Thus we get that

$$\begin{aligned}
T^{r-1}_{r0+1} &\geq (r - r1) + (r1 - r0) - TT(l, k') - 1 \\
&\geq DD_{\mathcal{B}}(i, k) + (\bar{k} - \bar{k'})PTO - TT(l, k') - 1 \\
&= GTO(k) + (g_i - g_k - 1)GTO(0) + (\bar{k} - \bar{k'})PTO - TT(l, k') - 1 \\
&= GTO(k') + (g_i - g_k - 1)GTO(0) - TT(l, k') - 1 \\
&= GTO(k') + (g_i - g_{k'} - 1)GTO(0) - TT(l, k') - 1 \\
&= DD_{\mathcal{B}}(i, k') - TT(l, k') - 1.
\end{aligned}$$

(The fourth inequality follows because $GTO(k) + (\bar{k} - \bar{k'})PTO = GTO(k')$, and the fifth inequality follows because $g_k = g_{k'}$.)

Again, Lemma 2.5 implies that $DD_{\mathcal{B}}(i, k') - TT(l, k') = DD_{\mathcal{B}}(i, l)$, and hence

$$\begin{aligned}
T^{r-1}_{r0+1} &\geq DD_{\mathcal{B}}(i, l) - 1 \\
&= GTO(l) + (g_i - g_l - 1)GTO(0) - 1 \\
&\geq (n/\sqrt{t} + 3\sqrt{t} + (\sqrt{t} - \bar{l} - 1)PTO + 1) - 1 \\
&\geq n/\sqrt{t} + 3\sqrt{t}.
\end{aligned}$$

This completes the proof of the inductive step for part (a).

For part (b), suppose by way of contradiction that i becomes active at round r and process $l < i$ has not retired by round r. First assume i does not receive a **go ahead** message. If $g_l < g_i$, we get an immediate contradiction using the inductive step for part (a), since i becomes active at or after it becomes preactive. Otherwise, recall that k is the last process from which i receives an ordinary message before becoming active, and this message is received at round $r1$. If $k > l$, then since k became active before round r, the inductive hypothesis on part (b) implies that l must have retired before k became active and hence before round r. If $k = l$, then i becomes preactive only after $PTO - 1 = n/t + 1$ additional rounds in which it does not hear from l. We claim that l must have retired by that time. Because, otherwise, in this period l would have either performed a subchunk and informed its group or would have checkpointed a subchunk to a group $g \neq g_l$ and informed its group about the checkpoint. Since $g_i = g_l$, in both cases, i must have heard from l. Finally, if $l > k$, then before i becomes active it sends a **go ahead** message to l and waits for

a message from l for $PTO - 1$ additional rounds. Exactly as above, it follows again that since i does not receive any message from l, l must have retired.

Next assume i does get a **go ahead** message before becoming active. However, the same reasoning as above shows that by the time a process sends a **go ahead** message to process i, all processes $< i$ have retired, and we are done. □

Finally we can prove Theorem 2.8.

THEOREM 2.8. *In every execution of Protocol $\mathcal{B}$,*

(a) *at most $3n$ units of work are performed in total by the processes,*

(b) *at most $10t\sqrt{t}$ messages are sent,*

(c) *by round $3n + 8t$ all processes have retired.*

Proof. Parts (a) and (b) were argued in the beginning of section 2.4.

For part (c), let i be the last process that is active and consider its activation chain. We want to find the last round $r2$ in which i is active. It follows from Lemma 2.4 that the maximal number of useful rounds performed by any chain is $n + 3t$. Therefore, applying Lemma 2.6 with $k = 0$ we get that

$$n + 3t \geq T_1^{r2}(i) \geq r2 - 1 - TT(i, 0) + 1.$$

Thus

$$\begin{aligned}
r2 &\leq n + 3t + TT(i, 0) \\
&\leq n + 3t + TT(t - 1, 0) \\
&= n + 3t + (\sqrt{t} - 1)GTO(0) + (\sqrt{t} - 1)PTO \\
&= n + 3t + (\sqrt{t} - 1)(n/\sqrt{t} + 3\sqrt{t} + (\sqrt{t} - 1)(n/t + 2) + 1) + (\sqrt{t} - 1)(n/t + 2) \\
&\leq n + 3t + (\sqrt{t} - 1)(n/\sqrt{t} + 3\sqrt{t} + \sqrt{t}(n/t + 2) + 1) \\
&= n + 3t + (\sqrt{t} - 1)(n/\sqrt{t} + 3\sqrt{t} + n/\sqrt{t} + 2\sqrt{t} + 1) \\
&\leq n + 3t + \sqrt{t}(2n/\sqrt{t} + 5\sqrt{t}) \\
&\leq 3n + 8t. \qquad \square
\end{aligned}$$

3. An algorithm with effort $O(n + t \log t)$. In this section we prove that the effort of $O(n + t\sqrt{t})$ obtained by the previous protocols is not optimal, even for work-optimal protocols. We construct another work-optimal algorithm, Protocol $\mathcal{C}$, that requires only $O(n + t \log t)$ messages (and a variant that requires only $O(t \log t)$ messages), yielding a total effort of $O(n + t \log t)$. As is the case with Protocols $\mathcal{A}$ and $\mathcal{B}$, at most one process is active at any given time. However, in Protocol $\mathcal{C}$ it is not the case that there is a predetermined order in which the processes become active. Rather, when an active process fails, we want the process that is currently *most knowledgeable* to become the new active process. As we shall see, which process is most knowledgeable after an active process i fails depends on how many units of work i performed before failing. As a consequence, there is no obvious variant of Protocol $\mathcal{C}$ that works in the model with asynchronous processes and a failure detector.

Roughly speaking, Protocol $\mathcal{C}$ strives to "spread out" as uniformly as possible the knowledge of work that has been performed and the processes that have crashed. Thus each time the active process, say, i, performs a new unit of work or detects a failure, i tells this to the process j it currently considers least knowledgeable. Then process j becomes as knowledgeable as i, so after performing the next unit of work (or detecting another failure), i tells the process it now considers least knowledgeable about this new fact.

The most naïve implementation of this idea is the following: process 0 begins by performing unit 1 of work and reporting this to process 1. It then performs unit 2 and reports units 1 and 2 to process 2, and so on, telling process i mod t about units 1 through i. Note that at all times, every process knows about all but at most the last t units of work to be performed.

If process 0 crashes, we want the most knowledgeable alive process—the one that knows about the most units of work that have been done—to become active. (If no process alive knows about any work, then we want the highest numbered alive process to become active.) It can be shown that this can be arranged by setting appropriate deadlines. Moreover, the deadlines are chosen so that at most one process is active at a given time. The most knowledgeable process then continues to perform work, always informing the least knowledgeable process.

The problem with this naïve algorithm is that it requires $O(n + t^2)$ work and $O(n + t^2)$ messages in the worst case. For example, suppose that process 0 performs the first $t-1$ units of work, so that the last process to be informed is process $t-1$, and then crashes. In addition, $t/2 + 1, \ldots, t - 1$ crash. Eventually process $t/2$, the most knowledgeable nonretired process, will become active. However, process $t/2$ has no way of knowing whether process 0 crashed just after informing it about work unit $t/2$, or process 0 continued to work, informing later processes (who must have crashed, for otherwise they would have become active before process $t/2$). Thus process $t/2$ repeats work units $t/2 + 1, \ldots, t - 1$, again informing (retired) processes $t/2 + 1, \ldots, t - 1$. Suppose process $t/2$ crashes after performing work unit $t - 1$ and informing process $t-1$. Then process $t/2-1$ becomes active and again repeats this work. If each process $t/2 - 1, t/2 - 2, \ldots, 1$ crashes after repeating work units $t/2 + 1, \ldots, t - 1$, then $O(t^2)$ work is done, and $O(t^2)$ messages are sent. (A slight variant of this example gives a scenario in which $O(n + t^2)$ work is done, and $O(n + t^2)$ messages are sent.)

To prevent this situation, a process performs failure detection before proceeding with the work. The key idea here is that we treat failure detection as another type of work. This allows us to use our algorithm recursively for failure detection. Specifically, fault detection is accomplished by polling a process and waiting for a response or a timeout. The difficulty encountered by our approach is that, in contrast to the real work, the set of faulty processes is dynamic, so it is not obvious how these processes can be detected without sending (wasteful) polling messages to *nonfaulty* processes. In fact, in our algorithm we do not attempt to detect all the faulty processes, only enough to ensure that not too much work is wasted by reporting work to faulty processes.

3.1. Description of the algorithm. For ease of exposition we assume t is a power of 2. Again, the processes are numbered 0 through $t - 1$, and the units of work are numbered 1 through n. Although our algorithm is recursive in nature, it can more easily be described when the recursion is unfolded. Processing is divided into $\log t$ levels, numbered 1 to $\log t$, where level $\log t$ would have been the deepest level of the recursion had we presented the algorithm recursively. In each level, the processes are partitioned into groups as follows. In level h, $1 \le h \le \log t$, there are $t/(2^{\log t - h + 1})$ groups of size $2^{\log t - h + 1}$. Thus in level $\log t$, there are $t/2$ groups of size 2, in level $\log t - 1$ there are $t/4$ groups of size 4, and so on, until level 1, in which there is a single group of size t. Let $s_h = 2^{\log t - h + 1}$ denote the size of a group at level h. The first group of level h contains processes $0, 1, \ldots, s_h - 1$, the next group contains processes $s_h, s_h + 1, \ldots, 2s_h - 1$, and so on. Thus each group of level $h < \log t$ contains two groups of level $h + 1$. Note that each process i belongs to $\log t$ groups, exactly one on

each level. We let G_h^i denote the level h group of process i.

Initially process 0 is active. When process i becomes active, it performs fault detection in its group at every level, beginning with the highest level and working its way down, leaving level h as soon as it finds a nonfaulty process in G_h^i. Once fault detection has been completed on G_1^i, the set of all processes, process i begins to perform real work. Thus we sometimes refer to the actual work as G_0, or level 0, and the fault detection on level h as *work on level h*. For each $1 \leq h \leq \log t$, each time it performs a unit of work on G_{h-1}^i, process i reports that work to some process in G_h^i. (Observe that the above protocol requires at least n messages. However, it will later become clear that modifying this protocol so that when a process performs work on G_0 it reports only each time it completes n/t units of work will immediately give a work-optimal protocol that requires only $O(t \log t)$ messages.)

A *unit of fault detection* is performed by sending a special message "Are you alive?" to one process and waiting for a reply in the following round. An *ordinary* message informs a process at some level h, $1 \leq h \leq \log t$, of a unit of (real or fault-detection) work at level $h-1$. As we shall see, an ordinary message also carries additional information. These two are the only types of messages sent by an active process. As before, a process that has crashed or terminated is said to be *retired*. An inactive nonretired process only sends responses to "Are you alive?" messages.

Each process i maintains a list F_i of processes known by i to be retired. It also maintains an array of pointers, POINT_i, indexed by group name. Intuitively, $\text{POINT}_i[G_0]$ is the successor of the last unit of work known by i to have been performed (and therefore this is where i will start doing work when it becomes active). For $h \geq 1$, $\text{POINT}_i[G_h^j]$ contains the successor (according to the cyclic order in G_h^j, which we define precisely below) of the last process in G_h^j known by i to have received an ordinary message from a process in G_h^j that was performing (real or fault-detection) work on G_{h-1}^j. We call $\text{POINT}_i[G_h^j]$ *process i's pointer into* G_h^j. Process i's moves are governed entirely by the round number F_i and pointers into its own groups (i.e., pointers into groups G_h^i). Associated with each pointer $\text{POINT}_i[G]$ is a round number $\text{ROUND}_i[G]$ indicating the round at which the last message known to be sent was sent (or, in the case of G_0, when the last unit of work known to be done was done). Initially, $\text{POINT}_i[G_0] = 1$, $\text{POINT}_i[G_h^j]$ is the lowest-numbered process in $G_h^j - \{i\}$, and $\text{ROUND}_i[G_0] = \text{ROUND}_i[G_h^j] = 0$. We occasionally use $\text{ROUND}_i[G](r)$ to denote the value of $\text{ROUND}_i[G]$ at the beginning of round r; we similarly use $F_i(r)$ and $\text{POINT}_i[G](r)$.

The triple $(F_i, \text{POINT}_i, \text{ROUND}_i)$ is the *view* of process i. We also define the *reduced view* of process i to be $\text{POINT}_i[G_0] - 1 + |F_i|$; thus i's reduced view is the sum of the number of units of work known by i to be done and the number of processes known by i to be faulty. A process includes its view whenever it sends an ordinary message. When process i receives an ordinary message, it updates its view in light of the new information received. Note that process i may receive information about one of its own groups from a process not in that group. Similarly, it may pass to another process information about a group in which the other process is a member but to which i does not belong.

Let G_h^i be any group as described above, where the process numbers range from x to $y = x + |G_h^i| - 1$. There is a natural fixed cyclic order on the group, which we call the *cyclic order*. Process i sends messages to members of G_h^i in *increasing order*. By this we mean according to the cyclic order but skipping itself and all processes in F_i. Let $j \neq i$ be in G_h^i. Then j's *i-successor in* G_h^i is j's nearest successor in the

cyclic ordering that is not in $\{i\} \cup F_i$. We omit the i in "i-successor," as well as the name of the group in which the successor is to be determined, when these are clear from the context.

When process i first becomes active it searches for other nonretired processes as follows. For each level h, starting with $\log t$ and going down to 1, process i polls group G_h^i, starting with $\text{POINT}_i[G_h^i]$, by sending an "Are you alive?" message. If no answer is received, it adds this process to F_i. If $h < \log t$, process i sends an ordinary message reporting this newly detected failure to $\text{POINT}_i[G_{h+1}^i]$, sets $\text{POINT}_i[G_{h+1}^i]$ to its i-successor in G_{h+1}^i, and sets $\text{ROUND}_i[G_{h+1}^i]$ to the current round number. Process i repeats these steps until an answer is received or $G_h^i \setminus \{i\} \subseteq F_i$. It then enters level $h-1$ and repeats the process. Note that if no reply was received, then although the pointer into G_h^i does not change, the successor in G_h^i of $\text{POINT}_i[G_h^i]$ does change, because the successor function takes into account F_i, which has changed.

Level 0 is handled similarly to levels 1 through $\log t - 1$, but the process performs real work instead of polling and increases the work pointer after performing each unit of work. If $\text{POINT}_i[G_0] = n+1$ then process i halts, since in this case all the work has been completed. This completes the description of the behavior of an active process. The code for an active process appears in Figure 3.

```
1.  h := log t;
2.  while h > 0 do
3.          DONE := FALSE;
4.          while ¬DONE do
5.                  Send "Are you alive?" to POINT_i[G_h^i];
6.                  if no response
7.                  then   add POINT_i[G_h^i] to F_i;
8.                         if h ≠ log t
9.                         then   send ordinary message to point_i[G_{h+1}^i];
10.                               ROUND_i[G_{h+1}^i] := current round;
11.                               POINT_i[G_{h+1}^i] := successor(POINT_i[G_{h+1}^i]);
12.                        if G_h^i − F_i ≠ {i}
13.                        then   POINT_i[G_h^i] := successor(POINT_i[G_h^i]);
14.                        else   DONE := TRUE;
15.                 else   (i.e., response received) DONE := TRUE;
16.         h := h − 1;

    {Process level 0 (real work):}
17. while POINT_i[G_0] ≤ n do
18.         Perform work unit POINT_i[G_0];
19.         Send an ordinary message to POINT_i[G_1^i];
20.         ROUND_i[G_1^i] := current round;
21.         POINT_i[G_1^i] := successor(POINT_i[G_1^i]);
22.         POINT_i[G_0] := successor(POINT_i[G_0])
```

FIG. 3. *Code for active process i in Protocol $\mathcal{C}$.*

At any time in the execution of the algorithm, each inactive nonretired process i has a *deadline*. We define $D(i,m)$ to be the number of rounds that process i waits

from the round in which it first obtained reduced view m until it becomes active:

$$D(i,m) = \begin{cases} K(n+t-m)2^{n+t-1-m} & \text{if } m \geq 1, \\ K(t-i)(n+t)2^{n+t-1} & \text{otherwise,} \end{cases}$$

where $K = 5t + 2\log t$. As we show below (Lemma 3.2), K is an upper bound on the number of rounds that any process needs to wait before first hearing from the active process. (More formally, if j becomes active at round r and is still active K rounds later, then by the beginning of round $r + K$ all processes that are not retired will have received a message from j.) All our arguments below work without change if we replace K by any other bound on the number of rounds that a process needs to wait before first hearing from the active process. This observation will be useful later, when we consider a slight modification of Protocol $\mathcal{C}$.

If process i receives no message by the end of $D(i,0) - 1$, then it becomes active at the beginning of round $D(i,0)$. Otherwise, if at round r it receives a message based on which it obtains a reduced view of m, and if it receives no further messages by the end of round $r + D(i,m) - 1$, it becomes active at the beginning of round $r + D(i,m)$. This completes the description of the algorithm.

3.2. Analysis and proof of correctness.

LEMMA 3.1. *In every execution of Protocol $\mathcal{C}$ in which there are no more than $t-1$ failures, the work is completed.*

Proof. By assumption, one of the processes is correct, say, i. At some point process i will become active, since once every other process has retired process i will not extend its deadline. It is straightforward from inspection of the algorithm that at any time during the execution of the algorithm $\text{POINT}_i[G_0] = w$ if and only if the first $w - 1$ units of work have been performed and that when it becomes active process i performs all units of work from $\text{POINT}_i[G_0]$ through n. □

The next lemma shows that our choice of K has the properties mentioned above.

LEMMA 3.2. *If j is active at round r and is not retired by round $r + 5t + 2\log t$, then all processes that are not retired will receive a message from j before the beginning of $r + 5t + 2\log t$.*

Proof. It is immediate from the description of the algorithm that all nonretired processes have received a message from j by the time it has performed t units of work (at level G_0) after round r. Thus we compute an upper bound on the time it takes for j to perform t units of work starting at round r. In the worst case, j has just become active at the beginning of round r, and must do failure detection before reaching level G_0 and doing work. While doing this failure detection, j sends "Are you alive?" messages to at most $t + \log t$ processes (the extra $\log t$ is due to the fact that at each level, it may send one "Are you alive?" message to a process that is alive but crashes later, while j is doing failure detection on a larger group). After discovering a failure, process j sends an ordinary message; thus it sends at most t ordinary messages. Each message sent takes up one round; in addition, process j waits one round for a response after each "Are you alive?" message. This means that j spends at most $3t + 2\log t$ rounds in levels $G^j_{\log t}, \ldots, G^j_1$. Clearly, j spends $\leq 2t - 1$ rounds working at level G_0 in the course of doing t units of work (since it sends an ordinary message between each unit of work). The required bound follows. □

If i received its last ordinary message from j at round r, we call other processes that received an ordinary message from j after i did *first-generation processes* (implicitly, with respect to i, j, and r). If i did not yet receive any ordinary messages, then

the first-generation processes (with respect to i and r) are those that received an ordinary message from a process with a number greater than i. We define kth-generation processes inductively. If we have defined kth generation, then the $(k+1)$st generation are those processes that receive an ordinary message from a kth-generation process. The *rank* of a process is the highest generation that it is in.

LEMMA 3.3. *Let i receive its last ordinary message from j at round r, let m be the reduced view of i after receiving this message, and let ℓ be a kth-rank process with respect to i, j, and r. Then, after ℓ receives its last ordinary message, its reduced view is at least $m + k$.*

Proof. The proof is an easy induction on k, since when a kth-rank process becomes active, it knows about everything its parent knew when it became active and at least one more piece of work or failure. □

We say process i *knows more* than process j at round r if $F_i(r) \supseteq F_j(r)$ and for all groups G, $\text{ROUND}_i[G](r) \geq \text{ROUND}_j[G](r)$. Note that if equality holds everywhere then intuitively the two processes are equally knowledgeable. We first show that our algorithm has the property that for any two inactive nonretired processes, one of them is more knowledgeable than the other, unless they both know nothing; that is, the knowledge of two nonretired processes is never incomparable. This is important so that the "most knowledgeable" process is well defined. Moreover, the knowledge can be quantified by the reduced view. Process i knows more than inactive process j if and only if the reduced view of i is greater than the reduced view of j. Finally, the algorithm also ensures that the active process is at least as knowledgeable as any inactive nonretired process.

LEMMA 3.4. *For every round r of the execution the following hold.*

(a) *If process i received an ordinary message from process j at round $r' < r$ and i is inactive and has not retired by the beginning of round r, then at the beginning of round r, no processes other than j and processes in the kth generation with respect to i, j, and r', for some $k \geq 1$, know as much as i.*

(b) *Suppose process i received its last ordinary message at round r' (if i has received no ordinary messages then $r' = 0$) and m is i's reduced view after receiving this message. If i is not retired at the beginning of round $r = r' + D(i, m)$, and it receives no further ordinary messages before the beginning of round r, then at the beginning of round r no nonretired process knows more than i.*

(c) *At the beginning of round r, there is an asymmetric total order ("knows more than") on the nonzero knowledge of the nonretired processes that did not become active before round r, and the active process knows at least as much as the most knowledgeable among these processes. Moreover, for any two nonretired processes i and j, i knows more than j if and only if the reduced view of i is greater than the reduced view of j.*

(d) *At most one process is active in round r.*

Proof. The proof is by induction on r. The base case $r = 1$ is straightforward. Let $r > 1$ and assume that all parts of the lemma hold for smaller values of r. We prove it for r.

For part (a), observe that by the inductive hypothesis, (a) holds at the beginning of round $r - 1$. If no process is active in round $r - 1$, then no process's knowledge changes, so (a) holds at the beginning of round r as well. If process j' is active in round $r - 1$, then by part (c) of the inductive hypothesis, j' knows at least as much as i. Thus by part (a), it must be the case that j' is either j or some process in the kth generation with respect to i, j, and r' for some k (since, by assumption, i is not active at the beginning of round r). The only process whose knowledge changes

during round $r-1$ is one to which j' sends an ordinary message. It is immediate from the definition that this process must be in the kth generation with respect to i, j, and r for some k.

For part (b), we must consider two cases: $r' > 0$ and $r' = 0$. If $r' > 0$, let j be the process that wrote to i at r'. By part (a) we have that only j and processes in generation $k \geq 1$ with respect to i, j, and r' are as knowledgeable as i at any round in the interval $[r', r)$. By part (c), these can be the only processes active in this interval. Thus it suffices to argue that j and all processes of generation $k \geq 1$ with respect to i, j, and r' are retired by the beginning of round r. Since a reduced view is at most $n+t-1$, the highest rank a process could be in is $n+t-1$. We now argue that by the beginning of round $r' + D(i,m) > r' + (n+t-m)K + D(i,m+1) + \cdots + D(i,n+t-1)$ all processes of ranks 1 through $n+t-1$ have retired. More generally, we argue by induction on k that for every k with $1 \leq k \leq n+t-m-1$, by the beginning of round $s + (k+1)K + D(i,m+1) + \cdots + D(i,m+k)$, every process in ranks 1 to k has retired.

If $k = 1$, note that since i received an ordinary message from j at round r', by Lemma 3.2, every rank-1 process receives a message from j before round $r' + K$. By Lemma 3.3, the reduced view of any such process is at least $m+1$. Since i receives no message from j by round $r' + K$, it must be the case that j has retired by round $r' + K$. By definition, no rank-1 process can receive any messages at any round in $[r' + K, r)$ (otherwise it would have a rank higher than 1). Thus any rank-1 process i' became active before $r' + K + D(i', m+1)$, so by definition of K and the fact that $D(i', m+1) = D(i, m+1)$, i would have heard from i' before $r' + 2K + D(i, m+1)$. It is easy to check that $r' + 2K + D(i, m+1) < r' + D(i,m) = r$. Since i did not receive any messages by the beginning of round r, i' must have retired by then.

In general, consider a rank-$k+1$ process i' and assume inductively that every rank-k or lower process has retired by the beginning of round r. By definition of rank, i' received an ordinary message from a rank-k process, and, since these are all retired by round r, i' must have received this message before round r. By the inductive hypothesis on k, i' must have received its last ordinary message by the beginning of round $r' + (k+1)K + D(i, m+1) + \cdots + D(i, m+k) < r$ (again using the fact that $D(i,m) = D(i',m)$ if $m > 0$). By Lemma 3.3, the reduced view of i' when it received its last ordinary message before round r was at least $m+k+1$. Thus it must have become active before round $r' + (k+1)K + D(i, m+1) + \cdots + D(i, m+k+1)$ if it became active at all. Since i received no messages from i', it follows that i' must have retired before round $r' + (k+2)K + D(i, m+1) + \cdots + D(i, m+k+1) < r$. This completes the induction on k.

If $r' = 0$ we need the fact that $D(i,0) > (n+t)K + \max_{j>i}\{D(j,0)\} + D(i,1) + \cdots + D(i, n+t-1)$, which follows easily from the definitions. We claim that, for every $k \geq 0$, by round $(k+1)K + \max_{j>i}\{D(j,0)\} + D(i,1) + \cdots + D(i,k)$, every process in ranks 1 to k has retired. To see this, note that a rank-0 process j' (one with a higher number than i that received no messages) must have become active at round $D(j',0)$ and therefore must have retired by round $D(j',0) + K$. Thus a level-1 process received its last message by $\max_{j>i}\{D(j,0)\} + K$. We now proceed as in the case $r' > 0$.

To prove part (c), observe that the result is immediate from the inductive hypothesis applied to $r-1$ if there is no active process at the beginning of round $r-1$ (for in that case, no process's reduced view changes). Otherwise, suppose that j is active at the beginning of round $r-1$. If j does not send an ordinary message in round $r-1$, again the result follows immediately from the inductive hypothesis (since

no process's reduced view changes). If j does send an ordinary message to, say, process i, it is immediate that i and j know more at the beginning of round r than any other nonretired process, and that i's reduced view is greater than that of any other nonretired inactive process.

It remains to show part (d). Observe that the result is immediate if no process becomes active at round r. Now suppose that process i becomes active at the beginning of round r. We must show that no process that was active prior to round r is still active at the beginning of round r, and that no process besides i becomes active at round r. Let r' be the last round in which i received a message (as usual, if i received no messages prior to round r, then we take $r' = 0$) and suppose that m was i's reduced view at round r'. Then we must have $r = r' + D(i, m)$. From part (b), it follows that no nonretired process knows more than i at the beginning of round r. From part (c), it follows that any process that was active in the interval $[r', r)$ must know more than i. This shows that all processes that were active before round r must have retired by the beginning of round r. Suppose some other process i' becomes active at round r. We have just shown that i' does not know more than i. From part (c) it follows therefore that i' knows less than i. Thus part (b) provides a contradiction to the assumption that i' becomes active at round r. □

LEMMA 3.5. *The running time of the algorithm is at most* $tK(n+t)2^{n+t}$ *rounds.*

Proof. If process i's reduced view is m and it does not receive a message within $D(i, m)$ steps, then it becomes active. Each message that i receives increases its reduced view. Thus i becomes active in at most $D(i, 0) + \cdots + D(i, n + t - 1)$ rounds. Once active, arguments similar to those used in Lemma 3.2 show that it retires in at most $2n + 3t + 2\log t$ rounds. Thus the running time of the algorithm is at most $D(1, 0) + \cdots + D(1, n + t - 1) + 2n + 3t + 2\log t \leq tK(n + t)2^{n+t}$ rounds. □

The next lemma shows that an active process i does not send messages to retired processes that, because they were more knowledgeable than i, should have become active before i did. These messages are avoided because during fault detection i discovers that these processes have retired.

LEMMA 3.6. *If process* i' *gets an ordinary message at round* r' *from a process operating on group* $G^{i'}_{h-1}$ *and process* i *is active at the beginning of round* $r > r'$ *then the following hold.*

(a) *If* $\text{ROUND}_i[G^{i'}_h](r) \geq r'$, *then all processes in the interval* $[i', \text{POINT}_i[G^{i'}_h](r))$ *in the cyclic order on* $G^{i'}_h$ *are either retired by the beginning of round* r *or receive an ordinary message in the interval* $[r', \text{ROUND}_i[G^{i'}_h](r)]$ *from a process operating on* $G^{i'}_{h-1}$. *(If* $i' = \text{POINT}_i[G^{i'}_h](r)$, *then all processes in* $G^{i'}_h$ *are either retired by the beginning of round* r *or receive a message in the interval* $[r', \text{ROUND}_i[G^{i'}_h](r)]$ *from a process operating on* $G^{i'}_{h-1}$.*) Moreover, either* i*'s knowledge at the beginning of round* r *is greater than* i'*'s knowledge at the end of* r' *or* $i' \in F_i(r)$.

(b) *If* $\text{ROUND}_i[G^{i'}_h](r) < r'$, *then all processes in the interval* $[\text{POINT}_i[G^{i'}_h](r), i']$ *in the cyclic order on* $G^{i'}_h$ *are either retired by the beginning of round* r' *or receive a message in the interval* $(\text{ROUND}_i[G^{i'}_h](r), r']$ *from a process operating on* $G^{i'}_{h-1}$. *Moreover, all the processes in this interval are retired by the beginning of round* r, *and if* $G^i_h = G^{i'}_h$, *then all these processes will be in* F_i *by the time* i *begins to operate on* G^i_{h-1}.

Proof. We proceed by induction on r. The case $r = 1$ is vacuous. Assume that $r > 1$ and the result holds for $r - 1$. If $r' = r - 1$, then it must be the case that i' received its message from i, $\text{ROUND}_i[G^{i'}_h](r) = r - 1$, and $\text{POINT}_i[G^{i'}_h]$ is the successor

of i' in the cyclic order on $G_h^{i'}$, as computed by i in round $r-1$. It is easy to see that the result follows immediately in this case, because all processes in the interval $(i', \text{POINT}_i[G_h^{i'}])$ must be retired.

Suppose $r' < r-1$. If i is also active at round $r-1$, then the result is immediate from the inductive hypothesis unless $\text{ROUND}_i[G_h^{i'}]$ changes during round $r-1$. The description of the algorithm shows that $\text{ROUND}_i[G_h^{i'}]$ changes only if $G_h^i = G_h^{i'}$ and i is operating on group G_{h-1}^i, in which case $\text{ROUND}_i[G_h^{i'}]$ is set to $r-1$ at the end of round $r-1$ and $\text{POINT}_i[G_h^{i'}](r)$ is the successor of $\text{POINT}_i[G_h^{i'}](r-1)$ in the cyclic order on $G_h^{i'}$. In this case it is easy to see that the result follows from the inductive hypothesis; we leave details to the reader.

Thus we can assume without loss of generality that i becomes active at round r. Let $\text{ROUND}_i[G_h^{i'}](r) = r''$ and let $\text{POINT}_i[G_h^{i'}](r) = i''$. If $r'' \geq r'$, then it must be the case that i received a message from j at some earlier round s such that $\text{POINT}_j[G_h^{i'}](s) = \text{POINT}_i[G_h^{i'}](r)$ and $\text{ROUND}_j[G_h^{i'}](s) = \text{ROUND}_i[G_h^{i'}](r)$. Since we must have $r' \leq r'' = \text{ROUND}_j[G_h^{i'}](s) \leq s$, the result now follows from the induction hypothesis (using j and s instead of i and r).

It remains only to consider the case $r'' < r'$. Let $j' \in G_h^{i'}$ be the process that sent the ordinary message to process i' at round r', and suppose that j' became active at the beginning of round s'. We claim that we have the following chain of inequalities: $r'' \leq \text{ROUND}_{j'}[G_h^{i'}](s') < s' < r' < r$. Every inequality in this chain is immediate from our assumptions except the first one. Suppose that $\text{ROUND}_{j'}[G_h^{i'}](s') < r''$. From Lemma 3.4, it follows that $\text{ROUND}_k[G_h^{i'}](s') < r''$ for all processes k not retired by round s'. This means that no process not retired at s' knows that a message was sent at round r''. But at round $r > s'$, process i knows this fact (since by assumption $\text{ROUND}_i[G_h^{i'}](r) = r''$). This is impossible. Thus we must have $\text{ROUND}_{j'}[G_h^{i'}](s') \geq r''$. Note that $\text{POINT}_{j'}[G_h^{i'}](r') = i'$ by assumption. Thus, by the inductive hypothesis, all processes in the cyclic order on $G_h^{i'}$ in the interval $[i'', i')$ are either retired by the beginning of round s' or receive an ordinary message in the interval $[r'', \text{ROUND}_{j'}[G_h^{i'}](s')]$ from a process operating on $G_{h-1}^{i'}$. Since we also know that i' receives a message at round r' from a process operating on $G_{h-1}^{i'}$, this proves the first half of part (b). Since, by Lemma 3.4, all processes not retired by round r must be less knowledgeable than i at the beginning of round r, it follows from Lemma 3.4 that all the processes in the interval $[i'', i']$ in the cyclic order have in fact retired by round r. From the description of the algorithm, it follows that i will detect this fact before it starts operating on G_{h-1}^i. □

Observe that the algorithm treats "Are you alive?" messages as real work. Therefore, we refer to these messages as work unless stated otherwise. On the other hand, the ordinary messages are still referred to as messages.

Using Lemma 3.6, we can show that indeed effort is not wasted.

LEMMA 3.7. *At most $|G_h^i| + |G_{h-1}^i|$ units of work are done and reported to G_h^i by group G_h^i when operating on group G_{h-1}^i.*

Proof. Given i, h, and an execution e of Protocol $\mathcal{C}$, we consider the sequence of triples (x, y, z), with one triple in the sequence for every time a process $x \in G_h^i$ sends an ordinary message reporting a unit of work $y \in G_{h-1}^i$ to a process $z \in G_h^i$, listed in the order that the work was performed. We must show that the length of this sequence is no greater than $|G_{h-1}^i| + |G_h^i|$.

We say that a triple (x, y, z) is *repeated* in this sequence if there is a triple (x', y, z') later in the sequence where the same work unit y is performed. Clearly there are at

most $|G_{h-1}^i|$ nonrepeated triples in the sequence, so it suffices to show that there are at most $|G_h^i|$ repeated triples. To show this, it suffices to show that the third components of repeated triples (denoting which process was informed about the unit of work) are distinct. Suppose, by way of contradiction, that there are two repeated triples (x_1, y_1, z_1) and (x_2, y_2, z_1) with the same third component. Suppose that x_1 informed z_1 about y_1 in round r', and x_2 informed z_1 about y_2 in round r''. Without loss of generality, we can assume that $r' < r''$. Since (x_1, y_1, z_1) is a repeated triple, there is a triple (x_3, y_1, z_2) after (x_1, y_1, z_1) in the sequence. Let r_3 be the round in which x_3 became active, and let r_2 be the round in which x_2 became active. Let $s_j = \text{ROUND}_{x_j}[G_h^i](r_j)$ for $j = 2, 3$. By Lemma 3.6, if $s_2 \geq r'$, then either x_2's knowledge at the beginning of round s_2 is greater than z_1's knowledge at the end of r', or $z_1 \in F_{x_2}(r')$, and if $s_2 < r'$, then $z_1 \in F_{x_2}$ before x_2 starts operating on G_i^{h-1}. Since x_2 sends a message to z_1 while operating on G_i^{h-1}, it cannot be the case that $z_1 \in F_{x_2}$ before x_2 starts operating on G_i^{h-1}, so it must be the case that $s_2 \geq r'$ and x_2's knowledge at the beginning of round r_2 is greater than z_1's knowledge at the end of round r'. In particular, this means that x_2 must know that x_1 informed z_1 about y_1 at the beginning of r_2.

We next show that every process $x \in G_i^h$ that is active at some round r between r' and r_2 must know that x_1 informed z_1 about y_1 at the beginning of round r. For suppose not. Then, by Lemma 3.6, z_1 must have retired by the beginning of round r. Since, by Lemma 3.4, x is the most knowledgeable process at the beginning of round r, it follows that no process that is not retired knows that z_1 was informed about y_1. Thus there is no way that x_2 could find this out by round r_2.

It is easy to see that x_3 does not know that z_1 was informed about y_1 (for if it did, it would not repeat the unit of work y_1). Therefore, (x_3, y_1, z_2) must come after (x_2, y_2, z_1) in the sequence. Since $\text{POINT}_{x_2}[G_h^i](r'') = z_1$, and z_1 received an ordinary message from x_1 while operating on G_{h-1}^i at round r', it follows from Lemma 3.6 that between rounds r' and r'' every process in G_h^i that is not retired must receive an ordinary message. In particular, this means that x_3 must receive an ordinary message. Since all active processes between round r' and r'' know that z_1 was informed about y_1, it follows that x_3 must know it too by the end of round r''. But then x_3 would not redo y_1, giving us the desired contradiction. □

THEOREM 3.8. *In every execution of Protocol $\mathcal{C}$,*

(a) *at most $n + 2t$ units of real work are performed;*

(b) *at most $n + 8t \log t$ messages are sent;*

(c) *by round $t(5t + 2\log t)(n + t)2^{n+t}$, all processes have retired.*

Proof. Lemma 3.7 implies that the amount of real work units that are performed and reported to G_1 is at most $|G_0| + |G_1| = n + t$. In addition, each of the t processes may perform one unit without reporting it (because it retired immediately afterward). Summing the two, (a) follows.

For part (b), Lemma 3.7 implies that each $G_h^i, h > 0$, performs at most $|G_{h-1}^i| + |G_h^i|$ reported units of works when operating on G_{h-1}^i. (Here a unit is may be either a real work unit or an "Are you alive?" message.) Let $H = \{(h, i) : 1 \leq h \leq \log t,\ i \equiv 1 \pmod{2^{\log(t)+1-h}}\}$. Notice if we consider groups of the form G_h^i for $(h, i) \in H$ we count all the groups exactly once. The argument above tells us that the total number of reported units of work is

$$\sum_{(h,i)\in H} (|G_{h-1}^i| + |G_h^i|) \leq |G_0| + 3 \sum_{(h,i)\in H} |G_h^i|.$$

The reason for the factor of 3 is that if $h < \log t$, then $|G_h^i|$ occurs three times in the left-hand sum: once when considering the work performed by group G_h^i operating on G_{h-1}^i, once when considering the work performed by G_{h+1}^i when operating on G_h^i, and once when considering the work performed by G_{h+1}^{i+h} when operating on G_h^i. Clearly, the $|G_0|$ reported units performed on G_0 result in one message each, and the remaining ones result in two messages each (because then the unit itself is also a message). So the number of messages corresponding to reported units of work is at most

$$|G_0| + 6 \sum_{(h,i)\in H} |G_h^i| = n + 6t \log t.$$

In addition, the unreported units may result in messages. These consist both of "Are you alive?" messages sent by a process but not reported by it due to the fact it crashes or terminates immediately afterward, and of "Are you alive?" messages that were not reported because the recipient of the "Are you alive?" message responded. Each process in $G_h^i, h > 1$, can perform at most one such unreported unit when operating on G_{h-1}^i, and hence each group $G_h^i, h > 1$ performs no more than $|G_h^i|$ such units. In addition, we have to sum the answers of alive processes in $G_{h-1}^i, h > 1$ to "Are you alive?" message sent by G_h^i. Again, there are at most $|G_h^i|$ such answers. Finally, each process i sends messages to the other process in $G_{\log t}^i$ just before it starts operating, which together with the answers sums up to a total of no more than $2t$ messages. Therefore, the number of messages corresponding to unreported units of work is at most

$$2t + \sum_{(i,h)\in H,\, h>1} 2|G_h^i| = 2t \log t.$$

Summing the messages due to the reported units of work and the messages due to the unreported units of work, part (b) follows.

Part (c) is immediate from Lemma 3.5. □

We remark that we can improve the message complexity to $O(t \log t)$ (that is, remove the n term in (b) above) by informing processes in group G_1 after n/t units of work done at level G_0, rather than after every unit of work. This does not result in a significant increase in total work, but it does increase the time complexity. The increase in time complexity comes from an increase in K (the upper bound on the number of rounds, from the time the currently active process takes over, that any process needs to wait before first hearing from the active process). Formally, we have the following result.

COROLLARY 3.9. *Modifying Protocol $\mathcal{C}$ by informing processes in group G_1 after n/t units of work done at level G_0, rather than after every unit of work, yields a protocol that sends $O(t \log t)$ messages, performs $O(n+t)$ work, and terminates within $t(2n + 3t + 2 \log t)(n + t)2^{n+t}$ rounds.*

4. A time-optimal algorithm. All the algorithms we have considered so far are inherently sequential: there is only one process performing work at a time. If processes always have many (other) tasks that they can do, then the fact that all but one process is idle at a given time is not a great problem. On the other hand, time is certainly a critical element in many applications. In this section, we present an algorithm that aims to achieve maximum distribution of the workload among the processes. The algorithm is time optimal in the typical case where there are no faults,

and its performance degrades gracefully in the presence of faults. The basic ideas of this algorithm have been patented [9].

The idea of the algorithm is straightforward. We alternate *work phases* and *agreement phases*, until all the correct processes are sure that all the work has been done. In the first work phase, process j performs units of work $1+jn/t, \ldots, (j+1)n/t$ (we again assume for simplicity that n is a multiple of t) in the first n/t rounds. Process j starts the first agreement phase by broadcasting a message to all the other processes saying that it has done its work. In subsequent rounds, process j proceeds much as in eventual Byzantine agreement [10]: it broadcasts its current *view*—what work has been done, and which processes were alive at the end of the work phase, from its point of view. It continues to do so until (a) the set of processes that are currently alive, according to j's view, is the same in two consecutive rounds, or (b) it receives a message from some process i saying that i is done and containing i's view. In case (a) it takes as its *final view* its own view, while in case (b) it takes as its final view the view in i's message. In all cases, it then broadcasts a message saying it is done, together with its final view of which processes were alive at the end of the work phase and what work remains to be done, and terminates the phase.

Using the by-now standard techniques of [10], we can show that all the correct processes agree on their final view at the time when they terminate the phase, and a correct process is done by round $n/t + f + 2$, where f is the number of processes that are faulty during the agreement phase. Finally, all correct processes terminate at most one round after the first correct process terminates. We omit details here.

After process j terminates the first agreement phase, if, according to its final view, $n' > 0$ more work still needs to be done (perhaps because some process crashed before doing its allocated work) and $t' \geq t/2$ processes are still correct, then it starts the second work phase. It performs n'/t' units of work, with the work being divided among the correct processes according to their identification numbers.[3] After the work phase, there is an agreement phase, which is just like the first agreement phase, with one small change. Whereas in the first agreement phase if process j did not hear from process i during some round, then process j knew i was faulty, in later agreement phases, since i may be behind j by one step, j must allow i one round of grace before declaring it faulty. Similarly, in order to terminate, a process must have two consecutive rounds *after the grace round*, where its view of the set of currently alive processes is the same, or receive a message from another process saying it is done. We leave it to the reader to check that again at the end of the phase all correct processes agree that all the work has been performed or they agree on their final view and, in addition, that every correct process terminates no more than one round after the first correct process terminates.

We continue in this manner, provided no more than half of the processes that were correct at the beginning of a phase fail, until all correct processes are sure that all work has been done. If at any phase more than half the correct processes fail, we revert to another of our algorithms (for example, Protocol $\mathcal{A}$). We call this algorithm protocol $\mathcal{D}$; the code appears in Figure 4. In the code, we use the function $grade_S$, where S is a set of nonnegative integers; $grade_S(s) = k$ if there are k elements of S less than s.

[3]Since n' may not be divisible by t', a process might have to do $\lceil n'/t' \rceil$ work. We ignore this issue in the discussion, since its impact on complexity is negligible; however, the code takes it into account.

Main protocol

```
1.  S := {1, ..., n};      {S is the set of outstanding units of work}
2.  T := {0, ..., t − 1};  {T is the set of processes known to have been correct
                            at the end of the previous work phase}
3.  ROUND := 1;  {ROUND keeps track of whether to allow a grace round}
4.  while |S| > 0 do
5.        S' := {s ∈ S : grade_T(j)⌈|S|/|T|⌉ ≤ grade_S(s) < (grade_T(j) + 1)⌈|S|/|T|⌉};
6.        Perform work in S';
7.        Wait ⌈|S|/|T|⌉ − |S'| rounds;  {to make sure all processes spend equally
                                           long in this phase}
8.        S := S\S';  {update outstanding units of work}
9.        T' := T;
10.       Agree(ROUND);  {see code below}
11.       if |T'| > 2|T|  (i.e., more than half the processes alive at the end of the
                           previous work phase failed by the end of the current
                           work phase)
12.       then  perform work in S using Protocol A;
13.             S := ∅;
14.       ROUND := 0
```

Agree(ROUND)

```
1.  DONE := FALSE;
2.  U := T; {U keeps track of processes not known by j to be faulty}
3.  T := {j};
4.  while ¬DONE do
5.        U_j := U;  {save old value of U}
6.        Broadcast (j, S, T, DONE) to all processes in U;
7.        for i ∈ U_j do
8.              if received (i, S_i, T_i, DONE_i) and DONE_i = FALSE
9.              then  S := S ∩ S_i;
10.                   T := T ∪ T_i;
11.             if received (i, S_i, T_i, DONE_i) and DONE_i = TRUE
12.             then  S := S_i;
13.                   T := T_i;
14.                   DONE := TRUE;
15.             if no message received from i and ROUND ≥ 1
16.             then  U := U/{i};
17.       if U = U_j and ROUND ≥ 1
18.       then  DONE := TRUE;
19.       ROUND := ROUND + 1;
20. Broadcast (j, S, T, DONE) to all processes in U
```

FIG. 4. *Protocol $\mathcal{D}$: code for process j.*

We now analyze Protocol $\mathcal{D}$. The analysis splits into two cases, depending on whether it is the case that for every phase no more than half the processes that are correct at the beginning of the phase are discovered to have failed during the phase.

A process p is *thought to be correct* at the beginning of phase i if $i = 1$ or $i > 1$ and p is in the final view of some process p' that decided in the phase $i-1$ agreement protocol. Note that in the latter case p is in the final view of all processes that complete the phase $i-1$ agreement protocol.

THEOREM 4.1. *In every execution of Protocol $\mathcal{D}$ in which at most f processes fail, the following hold.*

1. *If for each phase no more than half the processes that are thought to be correct at the beginning of the phase are discovered to have failed by the end of that phase, then*

(a) *at most $2n$ units of work are performed;*
(b) *at most $(4f+2)t^2$ messages are sent;*
(c) *by round $(f+1)n/t + 4f + 2$, all processes have retired.*

2. *If in some phase more than half the processes that are thought to be correct at the beginning of some phase are discovered to have failed by the end of the phase, then*

(a) *at most $4n$ units of work are performed;*
(b) *at most $(4f+2)t^2 + 9t\sqrt{t}/(2\sqrt{2})$ messages are sent;*
(c) *by round $(f+1)n/t + 4f + 2 + nt/2 + 3t^2/4$, all processes have retired.*

Proof. For part 1, an easy induction on k shows that by the end of phase k, no more than $n/2^k$ units of work remain to be done, and no more than $n + \cdots + n/2^{k-1}$ units of work have been done. It follows that at most $2n$ units of work are done altogether. (We remark that there is nothing special about the factor "half" in our requirement that we revert to Protocol $\mathcal{A}$ if more than half the processes that were correct at the beginning of the phase are discovered to have failed during the phase. We could have chosen any factor α; a similar proof would show that by the end of phase k, at most $\alpha^k n$ units of work remain to be done, and no more than $n + \cdots + \alpha^{k-1}n$ units of work have been done, so that no more than $n/(1-\alpha)$ units of work are done altogether. However, it follows from results of [8] that if we allow an arbitrary fraction of the processes to fail at every step and do not revert to Protocol $\mathcal{A}$, it is possible to construct an execution where f processes fail and $\Omega(n \log(f)/\log\log(f))$ units of work are done altogether. Indeed, it follows from the arguments in [8] that this result is tight; there is a matching upper bound.) Since each nonfaulty process broadcasts to all the other nonfaulty processes in each round of an agreement phase, at most t^2 messages are sent in each such round. If f_k is the number of failures discovered during the kth agreement phase, then the first agreement phase lasts at most $f_1 + 2$ rounds, while for $k > 1$, the kth agreement phase lasts at most $f_k + 3$ rounds, because of the grace round. Thus, altogether, the agreement phases last at most $f + 3a - 1$ rounds, where a is the number of agreement phases. Since $a \leq f + 1$, the agreement phases last at most $4f + 2$ rounds, and at most $(4f+2)t^2$ messages are sent. Finally, to compute an upper bound on the total number of rounds, it remains only to compute how many rounds are required to do the work (since we know the agreement phases last altogether at most $4f + 2$ rounds). Recall that at the end of phase k, at most $n/2^k$ units of work need to be done. Since no more than half the processes fail during any phase, at least $t/2^k$ processes are nonfaulty. Thus at most $(n/2^k)/(t/2^k) = n/t$ rounds are spent during each work phase doing work. Since there are at most $f + 1$ work phases, this gives the required bound on the total number of rounds.

For part (2), first observe that if we revert to Protocol $\mathcal{A}$ at the end of phase

k, then by our earlier observations it is known to the remaining processes that no more than $n/2^{k-1}$ units of work remain to be done, and no more than $(2-1/2^{k-1})n$ units of work have been done. It is also easy to see that at least $t/2$ processes are discovered as faulty. Moreover, by the bounds in part 1, at most $(4f+2)t^2$ messages have been sent and $(f+1)n/t+4f+2$ rounds have elapsed. Now applying Theorem 2.3, we see that at most $3n/2^{k-1}$ work is performed by protocol $\mathcal{A}$, no more than $9(t/2)\sqrt{t/2} = 9t\sqrt{t}/(2\sqrt{2})$ messages are sent, and $nt/2^k + 3t^2/4$ rounds are required. By taking $k=1$ (the worst case), we get the bounds claimed in the statement of the theorem. □

While the worst-case message complexity of this algorithm is significantly worse than that of our other algorithms, the time complexity is better (at least, if less than half the correct processes fail in each phase). More importantly, the situation is particularly good if no process fails or one process fails. If no process fails, then n units of work are done, the algorithm takes $n/t+2$ rounds, and $2t^2$ messages are sent. If one process fails, then we leave it to the reader to check that the algorithm requires at most $n/t+\lceil n/(t(t-1))\rceil+6$ rounds, has message complexity at most $5t^2$, and at most $n+n/t$ units of work are done.

As we mentioned in the introduction, it is easy to modify this algorithm to deal with a somewhat more realistic setting, where work is continually coming in to the system. Essentially, the idea is to run eventual Byzantine agreement periodically (where the length of the period depends on the size of the workload and on other features of the system). We omit further details here.

We can also cut down the message complexity in the case of no failures to $2(t-1)$, rather than $2t^2$, while still keeping the same work and time complexity. Instead of messages being broadcast during the agreement phase, they are all sent to a central coordinator, who broadcasts the results. If there are no failures, the agreement phase lasts two rounds, just as before. Dealing with failures is somewhat subtle if we do this though, so we do not analyze this approach carefully here.

5. Application to Byzantine agreement. One application of our algorithms is to Byzantine agreement. A Byzantine agreement protocol provides a means for n processes, at most t of which may be faulty, to agree on a value broadcast by a distinguished process called the *general* in such a way that all nonfaulty processes decide on the same value and, when the general is nonfaulty, they decide on the value the general sent. As in the rest of the paper, we restrict ourselves here to crash failures.

Consider the following Byzantine agreement algorithm. The algorithm proceeds in two stages: first, the general broadcasts its value to processes 0 to t; and then these $t+1$ processes employ one of our sequential algorithms to perform the "work" of informing processes 0 to $n-1$ about the general's value. So, performing one unit of work here means sending a message of the form "The general's value is x." To distinguish processes $0,\ldots,t$ from the others, we refer to them as the *senders*. A few more details are necessary to complete the description of the algorithm. First, throughout the algorithm, each process has a value for the general. Initially, the value is 0. If a process receives a message informing it about a value for the general different from its current value, it adopts the new value. Second, if the chosen work protocol is $\mathcal{C}$, then we modify it slightly so that each of its messages contains, in addition to its usual original contents, the current value the sender has for the general. Finally, at a predetermined time by which the underlying work protocol is guaranteed to have terminated, each process decides on its current value for the general.

Observe that processes $0, \ldots, t$ play two roles in the second stage of the Byzantine agreement algorithm: they both report the value of the general (as they do work) and are informed of this value (as work is performed on them by other senders).

We now prove the correctness of our Byzantine agreement algorithm. Obviously, if the general is correct, all processes will decide on its value. Since at least one of the $t+1$ senders is nonfaulty, it must be the case that a value is reported by a sender to every nonfaulty process. To show agreement, it suffices to show that there is no time at which two nonfaulty processes that have had a value reported to them by a sender have different (current) values. This, in turn, follows from the fact that if an active sender p reports a value v, and the sender that was active just before p reported $\bar{v}$, then at the time p becomes active, no value was reported to any nonfaulty process. Assume otherwise. Let p be the first active sender that violates this claim. Then p reports v for the general and the previous sender reported $\bar{v}$. Let q be the first sender that was active before p and reported $\bar{v}$; by construction, all senders that were active after q but before p reported $\bar{v}$.

By assumption, when q becomes active, no value was reported to any process that has not yet crashed. The choice of q guarantees that the only value that is reported from the time that q becomes active to the time that p becomes active is $\bar{v}$. It cannot be the case that a value was reported to p during this time, for then p's value when it becomes active would be $\bar{v}$, not v. In the case of Protocols $\mathcal{A}$ and $\mathcal{B}$, since work is done in increasing order of process number, it follows that no value is reported to any process with a higher number than p. (We remark that it is important here that a value is *not* included as part of the checkpoint messages in Protocols $\mathcal{A}$ and $\mathcal{B}$. Since checkpoint messages are broadcast, if a value were included, it is possible that a process that was active before p crashed while checkpointing to p; in this case, p may not have heard the value $\bar{v}$, and some process with a higher number than p may have heard it.) Moreover, the proof of correctness of Protocols $\mathcal{A}$ and $\mathcal{B}$ shows that all processes with a lower number than p must have crashed before p became active. Thus it follows that no value was reported to any nonfaulty process at the time p became active. In the case of Protocol $\mathcal{C}$, when p becomes active it is the most knowledgeable nonretired (and hence nonfaulty) process. Since for Protocol $\mathcal{C}$ we assume that the checkpointing messages include the value that was sent, no value can have been sent to any process that has not crashed.

Using Protocol $\mathcal{C}$, we get a Byzantine agreement protocol for crash failures that uses $O(n + t \log t)$ messages in the worst case, thus improving over Bracha's bound of $O(n + t\sqrt{t})$ [4]. Using $\mathcal{A}$ or $\mathcal{B}$, we match Bracha's message complexity, but our protocols are constructive, whereas Bracha's is not.

6. Conclusions. In this paper we have formulated the problem of performing work efficiently in the presence of faults. We presented three work-optimal protocols to solve the problem. One sends $O(t\sqrt{t})$ messages and takes $O(n + t)$ time, another requires $O(t \log t)$ messages at the cost of significantly greater running time, and the third optimizes on time in the usual case (where there are few failures). In particular, in the failure-free case, it takes $n/t + 2$ rounds and requires $2t^2$ messages.

There are numerous open problems that remain. For example, it would be interesting to see if message complexity and running time could be simultaneously optimized. It would also be interesting to prove a nontrivial lower bound on the message complexity of work-optimal protocols. Finally, note that by trying to optimize effort, the sum of work done and messages sent, we implicitly assumed that one unit of work was equal to one message. In practice, we may want to weight messages and work

differently. As long as the "weight" of a message is linearly related to the weight of a unit of work, then, of course, the complexity bounds for our algorithms continue to hold. However, if we weight things a little differently, then a completely different set of algorithms might turn out to be optimal. In general, it would be interesting to explore message/work/time tradeoffs in this model.

Acknowledgments. The authors are grateful to Vaughan Pratt for many helpful conversations, in particular for his help with the proof of Protocol $\mathcal{A}$. We also thank David Greenberg, Maurice Herlihy, and Serge Plotkin for their suggestions for improving the presentation of this work. The second author gratefully acknowledges the support of IBM.

REFERENCES

[1] R. J. Anderson and H. Woll, *Algorithms for the certified write-all problem*, SIAM J. Comput., 26 (1997), pp. 1277–1283. A preliminary version, containing other results, appears in Proc. 23rd ACM Symposium on Theory of Computing, 1991, pp. 370–380 (with the title *Wait free parallel algorithms for the union-find problem*).

[2] H. Attiya, A. Bar-Noy, and D. Dolev, *Sharing memory robustly in message-passing systems*, J. Assoc. Comput. Mach., 42 (1995), pp. 124–142. A preliminary version appears in Proc. 9th ACM Symposium on Principles of Distributed Computing, 1990, pp. 363–375.

[3] A. Bar-Noy and D. Dolev, *A partial equivalence between shared-memory and message-passing in an asynchronous fail-stop distributed environment*, Math. Systems Theory, 26 (1993), pp. 21–39.

[4] G. Bracha, Department of Computer Science, Cornell University, Ithaca, New York, July 1984, manuscript.

[5] M. F. Bridgland and R. J. Watro, *Fault-tolerant decision making in totally asynchronous distributed systems*, in Proc. 6th ACM Symposium on Principles of Distributed Computing, 1987, pp. 52–63.

[6] J. F. Buss, P. C. Kanellakis, P. L. Ragde, and A. A. Shvartsman, *Parallel algorithms with processor failures and delays*, J. Algorithms, 20 (1996), pp. 45–86.

[7] T. D. Chandra and S. Toueg, *Unreliable failure detectors for reliable distributed systems*, J. Assoc. Comput. Mach., 43 (1996), pp. 225–267. A preliminary version appears in Proc. 10th ACM Symposium on Distributed Computing, 1991, pp. 325–340.

[8] R. De Prisco, A. Mayer, and M. Yung, *Time-optimal message-efficient work performance in the presence of faults*, in Proc. 13th ACM Symposium on Principles of Distributed Computing, 1994, pp. 161–172.

[9] C. Dwork, J. Y. Halpern, and H. R. Strong, *Fault Tolerant Load Management System and Method*, U.S. Patent 5,513,354, 1996.

[10] D. Dolev, R. Reischuk, and H. R. Strong, *Early stopping in Byzantine Agreement*, J. Assoc. Comput. Mach., 34 (1990), pp. 720–741.

[11] Z. Galil, A. Mayer, and M. Yung, *Resolving message complexity of Byzantine agreement and beyond*, in Proc. 36th IEEE Symposium on Foundations of Computer Science, 1995, pp. 724–733.

[12] D. Gelernter and D. Kaminsky, *Supercomputing out of recycled garbage: Preliminary experience with Piranha*, in Proc. ACM International Conference on Supercomputing, 1992, pp. 417–427.

[13] P. C. Kanellakis and A. A. Shvartsman, *Efficient parallel algorithms can be made robust*, Distrib. Comput., 5 (1992), pp. 201–219.

[14] Z. Kedem, K. Palem, A. Raghunathan, and P. G. Spirakis, *Combining tentative and definite executions for very fast dependable parallel computing*, in Proc. 23rd ACM Symposium on Theory of Computing, 1991, pp. 381–389.

[15] Z. M. Kedem, K. V. Palem, and P. G. Spirakis, *Efficient robust parallel computations*, in Proc. 22nd ACM Symposium on Theory of Computing, 1990, pp. 138–148.

[16] C. U. Martel, A. Park, and R. Subramonian, *Work optimal asynchronous algorithms for shared memory parallel computers*, SIAM J. Comput., 21 (1992), pp. 1070–1099.

[17] J. F. Shoch and J. A. Hupp, *The Worm programs—early experience with a distributed computation*, Comm. Assoc. Comput. Mach., 25 (1982), pp. 95–103.

SIAM J. COMPUT.
Vol. 27, No. 5, pp. 1492–1513, October 1998
© 1998 Society for Industrial and Applied Mathematics
015

TIME–SPACE TRADEOFFS FOR UNDIRECTED *st*-CONNECTIVITY ON A GRAPH AUTOMATA*

JEFF A. EDMONDS†

Abstract. Undirected *st*-connectivity is an important problem in computing. There are algorithms for this problem that use $O(n)$ time and ones that use $O(\log n)$ space. The main result of this paper is that, in a very natural structured model, these upper bounds are not simultaneously achievable. Any probabilistic jumping automaton for graphs (JAG) requires either space $\Omega(\log^2 n/\log logn)$ or time $n^{(1+\Omega(1/\log\log n))}$ to solve undirected *st*-connectivity.

Key words. time–space tradeoffs, lower bounds, JAG graph, undirected *st*-connectivity

AMS subject classifications. 68Q05, 68Q15, 68Q25

PII. S0097539794277135

1. Introduction.

Graph connectivity is an important problem, both practically and theoretically. Practically, it is a basic subroutine to many graph theoretic computations. It the basic step in solving network flow optimization problems such as project scheduling and the matching of people to jobs. Graph connectivity is also important for computer networks and search problems. Theoretically, it has been studied extensively in a number of settings. Because the undirected version of the problem is complete for symmetric log-space and the directed version is complete for nondeterministic log-space, they are natural problems for studying these classes. The study of random walks on undirected graphs and deterministic universal traversal sequences has made the problem relevant to the issue of probabilism. In addition, the directed version was used by Karchmer and Wigderson to separate monotone NC_1 from NC_2. This paper proves time-space tradeoffs for undirected *st*-connectivity. (They apply to the harder problem of directed *st*-connectivity as well.) The importance of *st*-connectivity is discussed in more detail in Wigderson's beautiful survey [24].

The fastest algorithms for undirected graph *st*-connectivity are depth-first and breadth-first search [23]. These use linear time, i.e., $O(m+n)$ for an n node, m edge graph. However, they require $\Omega(n)$ space. Alternatively, this problem can be solved deterministically in $O\left(\log^{1.5} n\right)$ space and $n^{O(1)}$ time by traversing the graph using pseudo-random generators to describe a *universal traversal sequence* [17]. If nonuniformity is allowed, these bounds can be improved to $O(\log n)$ space and $O\left(n^4 \log n\right)$ time [10, 15]. If probabilism is allowed, random walks can traverse any component of the graph using $O(\log n)$ space and only $\Theta(mn)$ time [1]. More generally, Broder et al. [9] have exhibited a family of probabilistic algorithms that achieves a tradeoff between the time and the space of $S \cdot T \in m^2 \log^{O(1)} n$. This has been improved to $S \cdot T \in m^{1.5} n^{.5} \log^{O(1)} n$ [2]. A long-term goal is to prove a matching lower bound.

Proving lower bounds for a general model of computation such as a Turing machine is beyond the reach of the current techniques. Thus it is natural to consider

*Received by the editors November 2, 1994; accepted for publication (in revised form) August 1, 1996; published electronically May 21, 1998.
http://www.siam.org/journals/sicomp/27-5/27713.html

†Department of Computer Science, York University, North York, Ontario, Canada M3J 1P3. (jeff@cs.yorku.ca, http://www.cs.yorku.ca/jeff). This work was completed while the author was at the University of Toronto, Canada.

a "structured" model [8], whose basic operations are based on the structure of the graph, as opposed to being based on the bits in the graph's encoding. A natural structured model is the JAG introduced by Cook and Rackoff [11]. It has a set of states and a limited supply of labeled pebbles that it can either move from a node to an adjacent node ("walk") or move directly to a node containing another pebble ("jump"). The pebbles represent node names that a structured algorithm might record in its workspace and are useful for marking certain nodes temporarily, so that they are recognizable when other pebbles reach them. Walking represents replacing a node name by the name of a node that is adjacent to it in the input graph. Jumping represents copying a previously recorded node name [5]. The space of a JAG is defined to be $S = p \log_2 n + \log_2 q$, where p is the number of pebbles and q is the number of states, because it requires $\log_2 n$ bits to store the name of a node (i.e., the location of a pebble) and $\log_2 q$ bits to record the current state.

Although the JAG model is structured, it is not weak. In particular, it is general enough so that most known algorithms for graph connectivity can be implemented on it. For example, a JAG can perform depth-first or breadth-first search. It avoids cycling by leaving a pebble on each node when it first visits it. This uses $O(n \log n)$ space. Cook and Rackoff [11] show that the JAG model is powerful enough to execute an adaptation of Savitch's algorithm [20] for directed st-connectivity using only $O\left(\log^2 n\right)$ space. Poon [18] shows that Barnes's et al. [3] sublinear space, polynomial time algorithm for directed st-connectivity runs on a JAG as well.

Furthermore, Savitch [21] shows that if one allows the JAG the additional ability to move pebbles from each node i to node $i+1$ (for an arbitrary ordering of the nodes), then the model can simulate an arbitrary Turing machine on directed graphs. Beame et al. [5] show that even without this extra feature, JAGs can solve any undirected graph problem to within a polynomial factor as fast as Turing machines.

A number of space lower bounds have been obtained (even when an unbounded amount of time is allowed). Cook and Rackoff [11] prove a lower bound of $\Omega(\log^2 n / \log \cdot \log n)$ on the space required for a JAG to compute directed st-connectivity. This has been extended to randomized JAGs by Berman and Simon [6]. For undirected graph st-connectivity Cook and Rackoff [11] prove that $pq \in \omega(1)$ and Beame et al. [5] prove that if the pebbles are not allowed to jump, then $pq \in \Omega(n)$ even for simple 2-regular graphs. These proofs for undirected graphs show that, with a sublinear number of states, the model goes into an infinite loop. (This method does not work when there are a linear number of states, because then the JAG is able to count the number of nodes traversed.)

Tradeoffs between the number of pebbles p used and the amount of time needed for undirected graph st-connectivity have also been obtained. These results are particularly strong, because they do not depend on the number of states q. A *universal traversal sequence* is simply a JAG with an unlimited number of states but only one pebble. Borodin, Ruzzo, and Tompa [7] prove that on this model, undirected st-connectivity requires $\Omega\left(m^2\right)$ time. Beame et al. [5] extend this to $\Omega\left(n^2/p\right)$ for p pebbles on 3-regular graphs with the restriction that all but one pebble are unmovable. Thus, for this very weak version of the model, a quadratic lower bound on time×space has been achieved. Beame et al. [5] also prove that there is a family of $3p$-regular undirected graphs for which st-connectivity with $p \in o(n)$ pebbles requires time $\Omega(m \log(\frac{n}{p}))$, when the pebbles are unable to jump. The new result presented in this paper is the following.

THEOREM 1. *Any probabilistic JAG requires either space* $\Omega(\log^2 n/\log \log n)$ *or time* $n^{(1+\Omega(1/\log\log n))}$ *to solve undirected st-connectivity even for* 3*-regular graphs.*

This result improves upon the previous results in at least five ways: the lower bound on *time is larger*, all pebbles are allowed to *jump*, the *degree* of the graphs considered is constant, it applies to an *average case* input instead of just the worst-case input, and *probabilistic* algorithms are allowed.

The essential reason that tradeoffs arise between the time and the space required to solve a problem is that when the space is bounded the computation cannot store the results to all the previously computed subproblems and hence must recompute them over and over again. In order to prove a superlinear lower bound on the time to compute *st*-connectivity, it must be proved that certain subgraphs must be traversed many times. Although the first time the JAG traverses a particular subgraph may take many time steps, it can use its states to record the structure of the subgraph so that subsequent traversals can be performed more quickly. To deal with this situation, the lower bound is actually proved on a stronger model, called the *helper JAG*. In this model, the helper, after learning the input, is allowed to set the JAG's initial state and the initial position of the pebbles in the way that minimizes the computation time. The helper is able to communicate to the JAG at least as much information about the input as it could remember from a first traversal of a subgraph. In effect, a helper JAG lower bound is a bound on the time for a regular JAG to traverse the graph subsequent times.

The helper JAG lower bound is obtained by reducing *st*-connectivity to *st*-traversal and reducing the traversal problem to a game, referred to as *the helper-parity game*. The game characterizes the relationship between the number of bits of help (or foreknowledge) and the resulting traversal time.

The paper is structured as follows. Section 2 describes the helper parity game. Section 3 formally defines the probabilistic JAG model. Section 4 reduces the *st*-traversal problem to the *st*-connectivity problem. Section 5 defines the helper JAG. Section 6 describes the fly swatter graph and gives the complexity of its traversal. Section 7 describes a line of fly swatter graphs and compares its complexity with that of the helper-parity game. Section 8 describes how the input graph is recursively built. Section 9 provides the definitions of the time, pebbles, and cost used at a particular level of the recursion. This section does not provide a formal proof, but it does give some crucial intuition. Section 10 proves the main lemma by reducing the traversal problem to the helper parity game. Section 11 proves the main theorem from the main lemma. Finally, section 12 presents some strong intuition and possible techniques for improving the lower bound to the desired bound of $n^{2-o(1)}$.

2. The helper-parity game. The task of one instance of the parity game is to find a subset of the indexes on which the input vector has odd parity. This idea was introduced by Borodin, Ruzzo, and Tompa [7]. We extend this game by including a helper and multiple instances of the game. The helper is included in order to characterize what the JAG stores about a computation when a problem must be computed many times. Multiple instances of the game are included in order to decrease the number of help bits per game instances (the number of bits of information that the helper is allowed to give the player is bounded).

The helper-parity game with d game instances is defined as follows. There are two parties: a player and a helper. The input $\vec{\alpha}$ consists of d nonzero r bit vectors $\alpha_1, \ldots, \alpha_d \in \{0,1\}^r - \{0^r\}$, one for each of the d game instances. The helper sends the message $M(\vec{\alpha})$ to the player. It consists of a total of b bits about the d vectors. Then

the player repeatedly asks the helper parity questions. A parity question specifies one of the game instances $i \in [1 \ldots d]$ and a subset of the indexes $E \subseteq [1 \ldots r]$. The answer to the parity question is the parity of the input α_i at the indexes in this set, namely, $\bigoplus_{j \in E} [\alpha_i]_j$. The game is complete when the player has received an answer of 1 for each of the game instances. To simplify the game, the player is only charged one for the first question asked about a game instance and an additional one for any subsequent questions about the same game instance. Hence, the game can be thought of as the player repeatedly selecting a game instance i (in any order) and asking a single parity question E_i about that instance. If the parity is odd, the player is charged one for the game instance. Otherwise, he is charged two. Independent of the answer, the helper reveals the input α_i to the player. This is repeated until one question has been asked about each of the game instances. The cost is defined to be the average charge per game instance.

$$c_{\vec{\alpha}} = \frac{1}{d} \sum_{i \in [1 \ldots d]} \left\{ \begin{array}{ll} 1 & \text{if } \bigoplus_{j \in E_i} [\alpha_i]_j = 1 \\ 2 & \text{otherwise} \end{array} \right\}.$$

Without any help, the expected charge per game instance is 1.5. Lemma 1 below proves that if the amount of help is less than a fraction of a bit per game instance, then the player cannot do much better than this.

Defining the game so that the vector α_i is revealed after only one question is asked about it simplifies the proof of the *st*-connectivity lower bound without significantly affecting the bound obtained.

LEMMA 1. $\Pr_{\vec{\alpha}} [c_{\vec{\alpha}} < 1.5 - \epsilon] \leq 2^b e^{-(2\epsilon^2 - 2^{-r+1})d}$.

Proof. Fix a helper message $m \in \{0,1\}^b$. Randomly choose an input $\vec{\alpha}$ uniformly over all the inputs in $(\{0,1\}^r)^d$. We will consider what the player's protocol is given this message and this input, even if the helper does not send this message on this input. For $i \in [1 \ldots d]$, let x_i be the indicator variable specifying whether the question about the ith game instance had odd parity; i.e., $\bigoplus_{j \in E_i} [\alpha_i]_j = 1$. Clearly, these variables are independent and have probability $\frac{1}{2}$. The cost per game instance given the input $\vec{\alpha}$ and the actions of the player on message m are defined to be

$$c_{\langle \vec{\alpha}, m \rangle} = \frac{1}{d} \sum_{i \in [1 \ldots d]} \left\{ \begin{array}{ll} 1 & \text{if } x_i = 1 \\ 2 & \text{otherwise} \end{array} \right\}.$$

From this and Chernoff's bound [22], we get that

$$\Pr_{\vec{\alpha}} \left[c_{\langle \vec{\alpha}, m \rangle} < 1.5 - \epsilon \right] = \Pr \left[\sum_{i \in [1 \ldots d]} x_i \geq \left(\frac{1}{2} + \epsilon \right) d \right] \leq e^{-2\epsilon^2 d}.$$

Therefore, the cost is low for at most this fraction of vectors $\vec{\alpha} \in (\{0,1\}^r)^d$ or, equivalently, for at most $e^{-2\epsilon^2 d} 2^{rd}$ inputs $\vec{\alpha}$.

There are 2^b different helper messages m. For each message, the player's actions are different. Hence, the set of inputs on which the cost is low may be different for each message and in the worst case are disjoint. Hence, the number of inputs $\vec{\alpha}$ for which the cost is low when the player is given the correct help from the helper is at most $2^b e^{-2\epsilon^2 d} 2^{rd}$. The input $\vec{\alpha}$ is actually chosen randomly from $(\{0,1\}^r - \{0^r\})^d$. It

follows that

$$\Pr_{\vec{\alpha}}\left[c_{\vec{\alpha}} < 1.5-\epsilon\right] \le 2^b e^{-2\epsilon^2 d}\frac{2^{rd}}{\left(2^r-1\right)^d} \;=\; 2^b e^{-2\epsilon^2 d}\left(1-2^{-r}\right)^{-d}$$
$$\le 2^b e^{-2\epsilon^2 d} 2^{2^{-r+1}d} \;\le\; 2^b e^{-(2\epsilon^2-2^{-r+1})d}. \qquad \square$$

My thesis [12] presents some surprising results about various versions of this game. It turns out that the helper must give the player quite of few bits of help before significantly decreasing the number of parity questions the player must ask when playing a single game instance. However, the same number of bits of help will decrease the number of questions asked down to only two question per game instance, no matter how many game instances are being played simultaneously. However, the helper must provide an additional bit of help per game instance to decrease the number of questions asked per game instance below $2-\epsilon$.

This upper bound is best understood by not considering the message sent by the helper as being information about the input but as being random bits. With a few "random" bits, the worst-case complexity decreases to the point that it matches the expected complexity for a randomized protocol, which is two questions per game instance. The upper and lower bounds on the number of helper bits required match the work done by Impagliazzo and Zuckerman [14] on recycling random bits.

3. The probabilistic JAG model. A probabilistic JAG [11] is a finite automaton with p distinguishable pebbles and q states. The input to a JAG is an n vertex d regular undirected graph with two distinguished vertices s and t. For each vertex v, there is a labeling of the (half) edges emanating from v with d distinct labels. The set of labels used at different vertices is the same, but an edge can receive two different labels at its two endpoints. One of the pebbles is initially placed on the distinguished node t and other $p-1$ are placed on s.

The program of the JAG may depend nonuniformly on n and on the degree d of the graph. What the JAG does each time step depends on the current state, which pebbles coincide on the same vertices, which pebbles are on the distinguished vertices s and t, and the value $R \in \{0,1\}^*$ of some random bits. Based on this information, the automaton changes state and does one of the following two things. It either selects some pebble $P \in [1 \ldots p]$ and some label $l \in [1 \ldots d]$ and *walks* P along the edge with label l or it selects two pebbles $P, P' \in [1 \ldots p]$ and *jumps* P to the vertex occupied by P'. A JAG that solves st-connectivity enters an accepting state if and only if there is a path from s to t in the input graph.

The space of a JAG is defined to be $S = p \log_2 n + \log_2 q$, where p is the number of pebbles and q is the number of states, because it requires $\log_2 n$ bits to store the name of a node (i.e., the location of a pebble) and $\log_2 q$ bits to record the current state.

In this paper, the running time of a deterministic algorithm averaged over inputs according to a fixed input distribution is considered instead of considering the expected running time for a probabilistic algorithm on the worst-case input. According to Yao [25] this is sufficient.

4. Graph traversal. If it is possible for a pebble of a JAG to walk from s to t on a graph, then a graph is st-connected. However, a JAG can determine that the graph is st-connected in other ways. For example, suppose that at time T_0, pebble P_s is on vertex s, P_t is on t, and P_1 and P_2 are both on some third vertex v; at time $T' \in [T_0 \ldots T_1]$, P_s and P_1 are both on the same vertex v'; and at time $T'' \in [T_0 \ldots T_1]$,

P_t and P_2 are both on some other vertex v''. If these pebbles only walk along edges of the graph, then it follows that the graph is st-connected.

Additional complexity is caused by the pebbles being able to jump. A single pebble may not walk these paths from s to t. Instead, one pebble may walk part of the way. Another pebble may jump to this pebble and continue on the walk. In general, one cannot assume that a task is completed by a "specific pebble," because the pebbles are able to continually change places and each could complete some fraction of the task.

Such complex procedures for determining st-connectivity are captured by the *st-traversal* problem, which is formally defined as follows. Given a JAG computation on a graph G, the *traversal graph* H is defined as follows. For every vertex v of G and step $T \in [T_0 \dots T_1]$, let $\langle v, T\rangle$ be a vertex of H if and only if there is a pebble on vertex v at time T. Let $\{\langle u, T\rangle, \langle v, T+1\rangle\}$ be an edge of H if a pebble walks along edge $\{u, v\}$ in G during step T and let $\{\langle v, T\rangle, \langle v, T+1\rangle\}$ be an edge of H if there is a pebble that remains on vertex v during step T. We say that the JAG *traverses* the graph G from vertex s to vertex t during the time interval $[T_0 \dots T_1]$ if and only if there is an undirected path in H between $\langle s, T_s\rangle$ and $\langle t, T_t\rangle$ for some $T_s, T_t \in [T_0 \dots T_1]$.

In the example given above there is a traversal path composed of the four segments (see Figure 1): from $\langle s, T_0\rangle$ to $\langle v', T'\rangle$ following the movements of P_s, from $\langle v', T'\rangle$ to $\langle v, T_0\rangle$ following the movements of P_1 backward in time, from $\langle v, T_0\rangle$ to $\langle v'', T''\rangle$ following the movements of P_2, and from $\langle v'', T''\rangle$ to $\langle t, T_0\rangle$ following the movements of P_t backward in time. From the existence of this path, the JAG can deduce that s and t are connected.

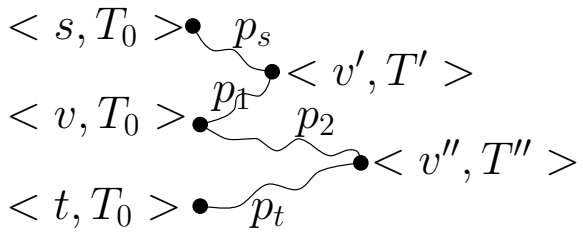


FIG. 1. *A path in the traversal graph of G.*

The st-connectivity decision problem and the problem of traversing from s to t are closely related. Let $\mathcal{F}$ be a family of connected graphs with distinguished nodes s and t. Let $G_{s,t} \cup G_{s',t'}$ be the graph composed of two identical disjoint copies of the graph $G \in \mathcal{F}$. Let $G_{s,t'} \cup G_{s',t}$ be the same except that one copy has the vertices s and t' and the other has s' and t. The first is st-connected and the second is not. Let $\mathcal{F}'$ be the family of graphs $\{G_{s,t} \cup G_{s',t'} \mid G \in \mathcal{F}\} \cup \{G_{s,t'} \cup G_{s',t} \mid G \in \mathcal{F}\}$.

LEMMA 2. *If a JAG solves st-connectivity (in T steps) with probability $\frac{1}{2} + \epsilon$ for input graphs uniformally chosen from $\mathcal{F}'$, then a similar JAG can perform st-traversal (in T steps) with probability 2ϵ for graphs uniformally chosen from $\mathcal{F}$.*

Proof. Suppose that there is a JAG that solves st-connectivity (in T steps) with probability $\frac{1}{2} + \epsilon$ for graphs uniformally chosen from $\mathcal{F}'$. We will say that the same JAG can perform st-traversal (in T steps) on a graph $G \in \mathcal{F}$, if on input $G_{s,t} \cup G_{s',t'}$ or $G_{s,t'} \cup G_{s,t'} \in \mathcal{F}'$ it either traverses from s to t or from s' to t'. By way of contradiction, suppose for a random graph $G \in \mathcal{F}$, the probability that it does this is strictly less than 2ϵ. Consider one of the $1 - 2\epsilon$ fractions of the graphs $G_{s,t} \cup G_{s',t'}$

on which it traverses neither from s to t nor from s' to t'. Consider the connected components of H. At each point in time, the pebbles can be partitioned based on which connected component of H they are in. A pebble can change which part of this partition it is in by jumping but not by traversing an edge. Since there is no path from s to t in H, the node $\langle s, T_s\rangle$, for any time step T_s, is in a different component than the nodes $\langle t, T_t\rangle$ for any time step T_t. Similarly, for all time steps $T_{s'}$ and $T_{t'}$, nodes $\langle s', T_{s'}\rangle$ and $\langle t', T_{t'}\rangle$ are in different components. If this JAG was given $G_{s,t'} \cup G_{s',t}$ as input instead, the connected components of H and the partitioning of the pebbles would be isomorphic. It follows that the computations on $G_{s,t} \cup G_{s',t'}$ and on $G_{s,t'} \cup G_{s',t}$ are identical. Therefore, for at least a half of the $1-2\epsilon$ fraction of the graphs being considered the JAG must give the wrong answer. This contradicts the assumption. □

By the definition of the *st*-traversal problem, all the pebbles are initially placed on s and t, and not within the graph. However, the proof of the lower bound uses an inductive argument for which the inductive step requires a somewhat stronger hypothesis than the main result, namely, that the result holds for traversing a subgraph (also with distinguished nodes s' and t') with an arbitrary initial placement of the pebbles and with an arbitrary initial state. If the subgraph initially contains pebbles within the subgraph or if the pebbles enter the subgraph via both the s' and the t' nodes, then the path through the traversal graph H may go forward and backward in time many times, as demonstrated in the above example. However, the following defines the type of traversals for which proving a lower bound is easier. Consider a subgraph with distinguished nodes s' and t' that initially contains no pebbles and that is built so that it can be entered by pebbles only via s' or t'. We will say that the subgraph has been *forward traversed* from s' to t' if it is traversed, yet, during the time period of the traversal, pebbles enter the subgraph via only one of the distinguished nodes s' or t' but not both. When this occurs, there must be a path from s' to t' or from t' to s' that proceeds only forward in time.

Consider a line of d subgraphs, the ith of which has distinguished nodes s_i and t_i, which are connected by the nodes s_i and t_{i+1} being the same node. The input graph will be many copies of this line of graphs. Consider a computation on this input graph starting with some initial placement of the pebbles and stopping when one of the lines of subgraphs has been traversed. Because the line is traversed, each of the subgraphs in the line must have been traversed. However, only some of these subgraphs would have been *forward traversed*. These need to be identified. Let $S_0 \subset [1 \dots d]$ consist of those i for which some copy of the line initially contains a pebble in its ith subgraph. Define $S_1 \subset [1 \dots d] - S_0$ to consist of those i such that the first time the ith subgraph in some line is traversed it is not forward traversed. These subgraphs do not initially contain pebbles; hence, when they are first traversed, pebbles must enter them via both the distinguished nodes s_i and t_i.

Claim 1. $|S_0 \cup S_1| \le 2p + 1$.

Proof. $|S_0| \le p$, because there are only p pebbles. Consider two indices i and $i' \in S_0$ such that $i < i'$ and there is no $i'' \in S_0$ strictly between $i < i'' < i'$. Consider two indexes j and j' such that $i < j < j' < i'$. If $j, j' \in S_1$, then pebbles would need to enter the jth subgraph via t_j and enter the j'th subgraph via $s_{j'}$. How did pebbles get in the line of subgraphs between these two nodes without pebbles first traversing the jth or the j'th subgraphs? Recall pebbles cannot jump to nodes not already containing pebbles. This is a contradiction. Hence, there can only be one $j \in S_1$ between i and i'. There can also be at most one $j \in S_1$ before first $i \in S_0$ and at most one after the last. Hence, $|S_1| \le |S_0| + 1$. □

For all $i \in [1 \ldots d] - (S_0 \cup S_1)$, the ith subgraph is forward traversed. The parameters will be defined so that $p \in o(d)$, so that most of the subgraphs need to be forward traversed.

5. The helper JAG. If the goal of the JAG is to compute the *st*-connectivity problem, then for any given input graph, a helper could tell the JAG the answer by communicating only one bit of help. On the other hand, we will show that many bits of help are required to significantly improve the time for the JAG to actually traverse from s to t in a certain class of graphs.

Having formally defined *st*-traversal, we are now able to formally define a helper JAG. It is the same as a regular JAG except that the helper, who knows the complete input, is able to set the initial state and pebble configuration in a way that minimizes the traversal time. Let $T\langle G, Q, \Pi\rangle$ be the time required for a regular JAG to traverse the input graph G starting in state $Q \in [1 \ldots q]$ and with $\Pi \in [1 \ldots n]^p$ specifying for each of the p pebbles which of the n nodes it is initially on. The time for the corresponding helper JAG to traverse G is defined to be $\min_{\langle Q,\Pi\rangle} T\langle G, Q, \Pi\rangle$. It is often easier to think of the helper giving the JAG $b = \log(qn^p)$ bits of information.

6. A fly swatter graph. The basic components of the input graphs defined in section 8 are the *fly swatter graphs* (see Figure 2). A fly swatter graph consists of two identical graphs with r *switches* between them. It is very similar to the squirrel cage graph defined in [5]. Each half consists of a path of length $\frac{h}{2}$, called the *handle*, and a swatting part. The swatting part consists of two parallel paths of length $r+1$ that are both connected to one end of the handle. The distinguished nodes s and t are located at the ends of the handles farthest from the swatting parts. Suppose the swatting part of one half of the graph contains the paths $u_0^0, u_0'^0, u_1^0, u_1'^0, \ldots, u_{r-1}'^0, u_r^0$ and $v_0^0, v_0'^0, v_1^0, v_1'^0, \ldots, v_{r-1}'^0, v_r^0$ and the swatting part of the other half contains the paths $u_0^1, u_0'^1, u_1^1, u_1'^1, \ldots, u_{r-1}'^1, u_r^1$ and $v_0^1, v_0'^1, v_1^1, v_1'^1, \ldots, v_{r-1}'^1, v_r^1$. Then the setting of the switches between the halves is specified by a nonzero vector $\alpha \in \{0,1\}^r$ as follows. For each $j \in [1 \ldots r]$, the jth switch consists of the two *cross-over edges* $\{u_j^0, v_j^{[\alpha]_j}\}$ and $\{u_j^1, v^{\overline{[\alpha]_j}}\}$. Note that if $[\alpha]_j = 0$, then the switch remains within the same half and if $[\alpha]_j = 1$, then the switch crosses over from one half to the other. (The notation $[\alpha]_j$ is used to denote the jth bit of the vector α. The notation α_i is reserved to mean the ith vector in a vector of vectors.) The extra nodes $u_i'^0$ and $v_i'^0$ are added so that the smallest square containing cross-over edges contains six edges.

Forward traversing from s to t in the fly swatter specified by α requires traversing a sequence of switches $E \in [1 \ldots r]^*$ for which the parity of the bits of α on the indexes in E is 1, i.e., $\bigoplus_{j \in E} [\alpha]_j = 1$. To be able complete this task, the JAG must be able to determine the parity of such a sequence E. There are two ways in which the JAG can ask a *parity question*. The lower bound will prove that these are the only ways in which the JAG is able to acquire the information about the input.

The first method of asking a parity question requires only one pebble but a great deal of time. The pebble enters the fly swatter via the distinguished node s (or t), traverses up the handle, through a sequence of switches $E \in [1 \ldots r]^+$, and back down the handle. While the pebble is inside the fly swatter, the JAG has no way of learning which half the pebble is in, because the two halves are indistinguishable. However, when the pebble reaches the bottom of the handle, the parity of the sequence is determined by whether the distinguished node s or t is reached. Each handle contains $\frac{h}{2}$ edges. Therefore, asking a parity question with one pebble requires the JAG to traverse at least h edges. This is illustrated in Figure 3 (a).

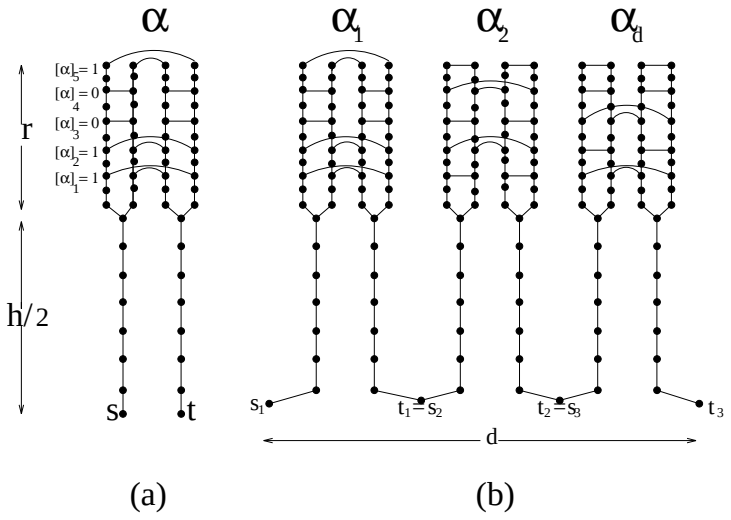


FIG. 2. *A fly swatter graph and a line of fly swatters.*

The second method requires two pebbles, one of which acts as a *marker* and the other of which traverses a sequence of switches E. The parity of the sequence is determined by whether the traversing pebble returns to the marker. For example, if a pebble starts at node u_2^0 and walks the edges labeled $\langle$ *switch,up,switch,down* $\rangle$, then one possible sequence of nodes for the pebble to follow is $u_2^0, v_2^0, v_3^0, u_3^0, u_2^0$ and another is $u_2^0, v_2^0, v_3^0, u_3^1, u_2^1$ depending on which switches in the graph are switched. Provided the JAG leaves a pebble as a marker at the node u_2^0, it can differentiate between these two sequences and learn whether $[\alpha]_2 \oplus [\alpha]_3$ is 0 or 1. This is illustrated in Figure 3 (b). Even though a parity question E (for example, the parity of all the bits in α) may require $\Theta(r)$ edges to be traversed, the lower bound will only charge the JAG for the traversal of at most two edges for a parity question using a marker.

If the JAG can only gain information about the input by asking parity questions in these two ways, then a JAG that solves *st*-connectivity for a fly swatter graph must be able to solve the following version of the parity game: The input to this game consists of a single nonzero vector $\alpha \in \{0,1\}^r$. The player, after receiving no help, asks parity questions. A parity question specifies a subset of the indexes $E \subseteq [1 \ldots r]$. The answer to the parity question is the parity of the input α at the indexes in this set, namely, $\bigoplus_{j \in E} [\alpha]_j$. The complexity is the number of questions asked before the player asks a parity question with answer 1.

Beame et al. [5] prove that, for this game, r questions must be asked in the worst case. The proof uses the fact that $\alpha \in \{0,1\}^r - \{0^r\}$ forms a vector space of dimension r. In this way, they prove that for one pebble, $\Theta(hr) = \Theta\left(n^2\right)$ time is needed. However, we are considering more than one pebble. With two pebbles the JAG can traverse the fly swatter graph in linear time.

In order to prove lower bounds when the JAG has more than one pebble, a more complex graph is needed. The fly swatter graph will be a subgraph of this more complex graph. In order to traverse this complex graph the JAG will have to traverse a particular fly swatter subgraph many times. Hence, on subsequent traversals of this fly swatter the JAG may have some precomputed information about it. This is modeled by a helper providing the JAG with this precomputed information.

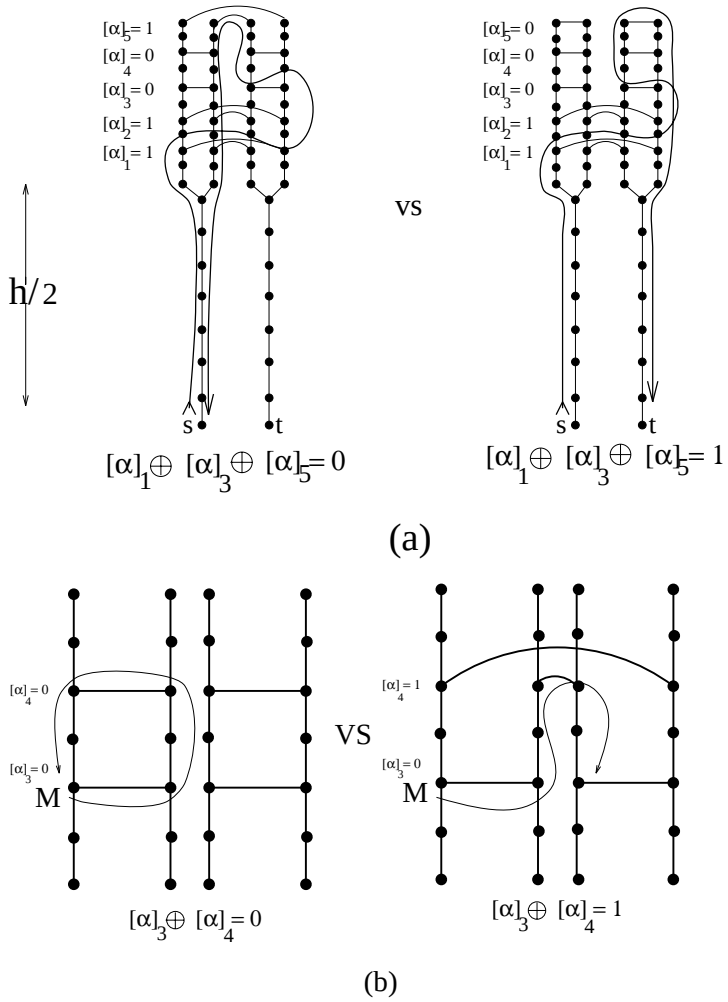


FIG. 3. *A parity question without and with a marker.*

7. The helper and a line of fly swatter graphs. The helper communicates information to the JAG about the input graph by specifying the initial state $Q \in [1 \ldots q]$ and location of each of the p pebbles. Hence, the amount of information that the helper is able to provide is limited to at most $b = \log(qn^p)$ bits. Only $\log r << b$ bits are required for the JAG to be able to traverse a fly swatter in linear time. However, b is not enough bits of help to simultaneously provide sufficient information about many fly swatter graphs. For this reason, we require the JAG to traverse a *line of d fly swatters*. Such a line is specified by the parameters $r \in O(1)$, $h \in O(1)$, $d \in \log^{O(1)} n$ and the vector of vectors $\vec{\alpha} = \langle \alpha_1, \ldots, \alpha_d \rangle \in (\{0,1\}^r - \{0^r\})^d$. The d graphs are connected by making the distinguished nodes t_i and s_{i+1} be the same node for $i \in [1 \ldots d-1]$. This is illustrated in Figure 2 (b).

The similarities between the parity game and the traversal of a line of fly swatter graphs should be clear, at least informally. Lemma 1 proves that if the JAG is only able to gain information by asking parity questions and $b << d$, then the average number of questions the JAG must ask is $(1.5 - \epsilon)\, d$. Therefore, if a marker is utilized in none

of the questions, then the number of edges traversed by the JAG is $h\,(1.5-\epsilon)\,d$, and if a marker is utilized in all of the questions, then $2\,(1.5-\epsilon)\,d$ edges are traversed. Note that without a marker, the time required is roughly a factor of $(1.5-\epsilon)$ larger than the number of edges. This factor is not very impressive, but its effect is magnified in a recursive construction.

8. The recursive fly swatter graphs. See Figure 4. Let $G\,(\vec{\alpha}_l)$ denote the line of fly swatters specified by the vector of vectors $\vec{\alpha}_l = \left\langle \alpha_{\langle l,1\rangle}, \ldots, \alpha_{\langle l,d\rangle} \right\rangle \in (\{0,1\}^r - \{0^r\})^d$. For each $l \geq 0$, we recursively define a graph. Define $G\,(\emptyset)$ to be a single edge. Define $G\,(\vec{\alpha}_1, \ldots, \vec{\alpha}_l)$ to be the graph $G\,(\vec{\alpha}_l)$ where each edge is replaced by a super edge consisting of a copy of $G\,(\vec{\alpha}_1, \ldots, \vec{\alpha}_{l-1})$. The node $s_{\langle l-1,1\rangle}$ is one end of the super edge and the node $t_{\langle l-1,d\rangle}$ is the other end. All the super edges in one level are the same. The family of recursive fly swatter graphs is $\left\{\, G\,(\vec{\alpha}_1, \ldots, \vec{\alpha}_L) \;\middle|\; \vec{\alpha}_1, \ldots, \vec{\alpha}_L \in (\{0,1\}^r - \{0^r\})^d \,\right\}$, where L is such that n is the total number of nodes. (Gadgets can be added to make the graph 3-regular without changing it significantly.) The graph $G\,(\vec{\alpha}_1, \ldots, \vec{\alpha}_l)$ contains $(h+10r)\,d$ copies of $G\,(\vec{\alpha}_1, \ldots, \vec{\alpha}_{l-1})$. Therefore, the total number of edges is at most $[(h+10r)\,d]^L$. The number of nodes n is approximately two thirds of the number of edges.

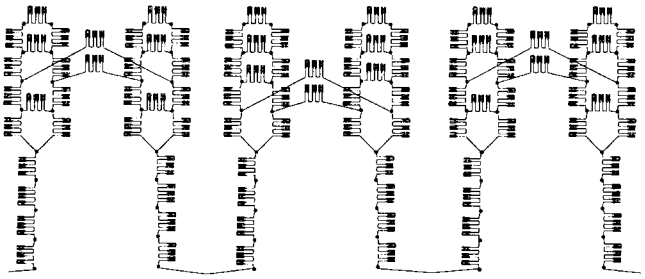


FIG. 4. *The recursive line of fly swatters graph. (Sorry, the cycles in the figure have length* 4 *instead of* 6.*)*

A crucial observation in understanding the complexity of traversing this graph is that a pebble can only be used as a marker in one recursion level at a time. To demonstrate this, consider $L = 2$ levels of recursion and $p = 2$ pebbles. If a parity question is asked about the top level vector $\vec{\alpha}_L$ by leaving a pebble as a marker, then only one pebble remains to traverse the sequence of super edges required by the question. Hence, these super edges, which are themselves lines of fly swatters, must be traversed with the use of only one pebble. Alternatively, if two pebbles are used to traverse each super edge, then there is "effectively" one pebble for the traversal of the top level. See Figures 3 (b) and 5.

An intuitive explanation of the lower bound can now be given. (A formal proof is provided in section 11.) There are p pebbles, hence at most $p-1$ markers. It follows that $L-p+1$ levels are traversed without the use of a marker. Note, as well, that the time to traverse a copy of $G\,(\vec{\alpha}_1, \ldots, \vec{\alpha}_l)$ is the number of super edges traversed in the lth level subgraph $G\,(\vec{\alpha}_l)$ multiplied by the time to traverse each super edge $G\,(\vec{\alpha}_1, \ldots, \vec{\alpha}_{l-1})$. Therefore, an estimation of the total time is the product of the number of super edges traversed at each of the L levels:

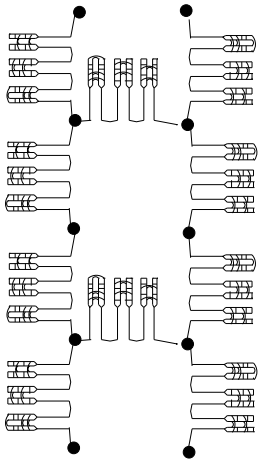


FIG. 5. *Ten super edges of a two level recursive fly swatter graph.*

$$
\begin{aligned}
T &\geq \left[h\,(1.5-\epsilon)\,d\right]^{L-p+1}\left[(1.5-\epsilon)\,d\right]^{p-1} \\
&= (1.5-\epsilon)^{L} \times (hd)^{L} \times h^{-p+1}. \qquad (8.1)
\end{aligned}
$$

The parameters are chosen as follows: $r, h \in \Theta(1)$, $d \in \log^{\Theta(1)} n$, and $L \in \Theta(\frac{\log n}{\log\log n})$. Then the first factor becomes $2^{\Omega(\frac{\log n}{\log\log n})}$, the second is close with n (assuming $r << h$), and the third is insignificant compared with the first assuming that p is a constant fraction of $\frac{L}{\log h} \in \Theta(\frac{\log n}{\log\log n})$. This is the bound claimed in Theorem 1.

9. The time, pebbles, and cost used at level *l*. The lower bound is proved by induction on the number of levels of recursion l. For each $l \in [1 \ldots L]$, we prove a lower bound on the cost to traverse some copy of $G(\vec{\alpha}_1, \ldots, \vec{\alpha}_l)$. Define $\vec{\gamma} = \langle \vec{\alpha}_1, \ldots, \vec{\alpha}_{l-1} \rangle$ and $\vec{\beta} = \langle \vec{\alpha}_{l+1}, \ldots, \vec{\alpha}_L \rangle$ so that $G(\vec{\alpha}_1, \ldots, \vec{\alpha}_L)$ and $G(\vec{\gamma}, \vec{\alpha}_l, \vec{\beta})$ denote the same graph. Think of $G(\vec{\gamma}, \vec{\alpha}_l)$ (the subgraph traversed) as a line of d fly swatters $G(\vec{\alpha}_l)$ with each of its super edges being a copy of $G(\vec{\gamma})$. The super edge $G(\vec{\gamma})$ does not need to be understood, because the induction hypothesis proves a lower bound on the time to traverse it. In the graph $G(\vec{\gamma}, \vec{\alpha}_l, \vec{\beta})$, there are many copies of $G(\vec{\gamma}, \vec{\alpha}_l)$. The graph $G(\vec{\beta})$ is called the *context* in which these copies of $G(\vec{\gamma}, \vec{\alpha}_l)$ appear.

Informally, the time required to traverse $G(\vec{\gamma}, \vec{\alpha}_l)$ is the number of super edges of $G(\vec{\alpha}_l)$ traversed multiplied by the time to traverse a super edge $G(\vec{\gamma})$. This fails to be true, because each traversal of a super edge may require a different amount of time. This difference in time is caused by the number of markers (pebbles) being used in the traversal and by the state and position of the pebbles before the traversal. Note that differences in traversal times are not caused by differences in the structure of the super edges, because they are all identical.

Let $Q \in [1 \ldots q]$ be a state of the JAG and let $\Pi \in [1 \ldots n]^p$ specify which node of $G(\vec{\gamma}, \vec{\alpha}_l, \vec{\beta})$ each of the p pebbles is on. Consider the JAG computation starting in the configuration described by Q and Π on the graph $G(\vec{\gamma}, \vec{\alpha}_l, \vec{\beta})$ until some copy of $G(\vec{\gamma}, \vec{\alpha}_l)$ is traversed. Define $T\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle, Q, \Pi\right]$ to be the number of time steps taken. This will be abbreviated to $T_{[l]}$ when the rest of the parameters are understood.

Define $p\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle, Q, \Pi\right]$ (abbreviated to $p[l]$) to be

$$p\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle, Q, \Pi\right] = \max_{T \in [1 \ldots T[l]]} \left[p + 1 - (\#\text{ copies of } G(\vec{\gamma}, \vec{\alpha}_l) \text{ containing pebbles at time } T)\right].$$

At no time during the interval does a copy of $G(\vec{\gamma}, \vec{\alpha}_l)$ contain more than $p[l]$ pebbles. For example, suppose there are two copies of $G(\vec{\gamma}, \vec{\alpha}_l)$ containing pebbles. One copy contains at least one pebble and, therefore, the other copy contains no more than $p + 1 - 2 = p - 1$ pebbles. Think of the "(# copies of $G(\vec{\gamma}, \vec{\alpha}_l)$ containing pebbles)" as the number of pebbles being used as "markers" in the graph $G(\vec{\beta})$. Essentially, no more than one of these pebbles is available to be used in the traversal of $G(\vec{\gamma}, \vec{\alpha}_l)$.

Define the *cost* incurred by the JAG in traversing a copy of $G(\vec{\gamma}, \vec{\alpha}_l)$ to be

$$w\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle\right] = \min_{\langle Q, \Pi \rangle} h^{p\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle, Q, \Pi\right]} T\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle, Q, \Pi\right].$$

The motivation for using $h^{p[l]}T[l]$ comes from (8.1). Since the average time $T[l]$ to traverse $G(\vec{\gamma}, \vec{\alpha}_l)$ can be estimated by $(1.5 - \epsilon)^l (hd)^l h^{-p[l]+1}$, the quantity $h^{p[l]}T[l]$ is essentially independent of the number of pebbles used. The reason for minimizing over $\langle Q, \Pi \rangle$ is that we are assuming the helper sets the initial JAG configuration in a way that minimizes the cost of the traversal. The actual bounds obtained (with high probability) are the following:

$$\begin{aligned} W[0] &= h, \\ W[l] &= W[l-1] \times h \times (d(1.5 - \epsilon) - 4p - 2) \\ &= [h \times (d(1.5 - \epsilon) - 4p - 2)]^l h. \end{aligned}$$

The remaining goal is to prove that with high probability (over randomly chosen inputs) the cost $w\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle\right]$ to traverse a copy of $G(\vec{\gamma}, \vec{\alpha}_l)$ is at least $W[l]$. Lemma 3 provides the inductive step for this proof.

10. Reducing the helper parity game to *st*-traversal. Suppose that there is a JAG algorithm for which the time to traverse a copy of $G(\vec{\gamma}, \vec{\alpha}_l)$ is only a small factor more than the time to traverse a copy of $G(\vec{\gamma})$. This means that the JAG is able to traverse the line of fly swatters $G(\vec{\alpha}_l)$ without traversing many of its super edges and hence without "asking" many parity questions about $\vec{\alpha}_l$. In effect, the JAG is able to play the helper parity game with parameters r, d, and $b = \log(qn^p)$, where r and d are the parameters defining the line of fly swatters and $\log(qn^p)$ is the space allocated to the JAG. This is captured in the following lemma.

LEMMA 3. *Given an algorithm for st-traversal whose cost on input $G(\vec{\gamma}, \vec{\alpha}_l, \vec{\beta})$ to traverse a subgraph at the $l-1$st and the lth levels are $w\left[l-1, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle\right]$ and $w\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle\right]$, we can produce a protocol for the helper parity game for which the number of questions asked per game instance on input $\vec{\alpha}_l$ is $c_{\vec{\alpha}_l}$ such that*

$$\begin{aligned} \Pr_{\vec{\alpha}_l}\left[c_{\vec{\alpha}_l} < 1.5 - \epsilon\right] \geq{}& \Pr_{\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \rangle}\left[w\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle\right] < W[l]\right] \\ &- \Pr_{\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \rangle}\left[w\left[l-1, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle\right] < W[l-1]\right]. \end{aligned}$$

COROLLARY 1. *Given a helper-JAG algorithm for st-traversal whose traversal time on input* $G(\vec{\alpha}_1, \ldots, \vec{\alpha}_L)$ *is* $T_{\langle \vec{\alpha}_1, \ldots, \vec{\alpha}_L \rangle}$ *and the helper parity game defined in Lemma* 3*,*

$$\Pr_{\langle \vec{\alpha}_1, \ldots, \vec{\alpha}_L \rangle} \left[T_{\langle \vec{\alpha}_1, \ldots, \vec{\alpha}_L \rangle} < \left[h \times (d(1.5 - \epsilon) - 4p - 2) \right]^L h^{-p+1} \right] \leq L \times \Pr_{\vec{\alpha}} \left[c_{\vec{\alpha}} < 1.5 - \epsilon \right].$$

Proof. The time and the cost incurred by the JAG to traverse the entire input graph are defined to be $T_{\langle \vec{\alpha}_1, \ldots, \vec{\alpha}_L \rangle} = T_{[L]}$ and $w_{[L]} = \min_{\langle Q, \Pi \rangle} h^{p_{[L]}} T_{[L]}$. The number of pebbles $p_{[L]}$ used in the traversal is at most the number of pebbles p the JAG has. It follows that if traversal time is small, i.e., $T_{[L]} < W_{[L]} h^{-p}$, then the cost is small, i.e., $w_{[L]} = h^{p_{[L]}} T_{[L]} < W_{[L]}$. It remains to prove that for $l \leq L$,

$$\Pr_{\langle \vec{\alpha}_1, \ldots, \vec{\alpha}_L \rangle} \left[w\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle \right] < W_{[l]} \right] \leq l \times \Pr_{\vec{\alpha}} \left[c_{\vec{\alpha}} < 1.5 - \epsilon \right].$$

The proof proceeds by induction on l. For the base case, $l = 0$, the subgraph $G()$ is a single edge requiring at least 1 time step for traversal by at least one pebble. This guarantees that with probability 1, $w_{[0]} = \min_{\langle Q, \Pi \rangle} h^{p_{[0]}} T_{[0]} \geq h = W_{[0]}$. The induction step follows easily using Lemma 3. □

Proof. Consider a fixed algorithm for *st*-traversal. In the helper-parity protocol defined below, the game helper learns what help to send and the game player learns what questions to ask by running this fixed JAG algorithm as it traverses a line of fly swatters at the lth level.

Both the *st*-traversal problem and the helper-parity game have the vector $\vec{\alpha}_l = \left\langle \alpha_{\langle l,1 \rangle}, \ldots, \alpha_{\langle l,d \rangle} \right\rangle \in (\{0,1\}^r - \{0^r\})^d$ as part of its input. However, the *st*-traversal problem has the additional inputs $\vec{\gamma}$ and $\vec{\beta}$. Therefore, these vectors are fixed to $\vec{\gamma}'$ and $\vec{\beta}'$ in a way that satisfies the property

$$\begin{aligned} &\Pr_{\vec{\alpha}_l} \left[w\left[l, \left\langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \right\rangle \right] < W_{[l]} \right] - \Pr_{\vec{\alpha}_l} \left[w\left[l-1, \left\langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \right\rangle \right] < W_{[l-1]} \right] \\ &\leq \Pr_{\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \rangle} \left[w\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle \right] < W_{[l]} \right] - \Pr_{\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \rangle} \left[w\left[l-1, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle \right] < W_{[l-1]} \right]. \end{aligned}$$

These vectors $\vec{\gamma}'$ and $\vec{\beta}'$ are known in advance to both the game helper and the game player.

The first thing that the game protocol must specify is the message $M(\vec{\alpha}_l)$ sent by the game helper on each input $\vec{\alpha}_l$. This message is defined to be $\left\langle Q_{\langle l, \vec{\alpha}_l \rangle}, \Pi_{\langle l, \vec{\alpha}_l \rangle} \right\rangle$, which specifies the configuration in which the JAG helper initially places the JAG when $G(\vec{\gamma}', \vec{\alpha}_l, \vec{\beta}')$ is the JAG's input graph. Note that the game helper only sends $\log(qn^p)$ bits, because this is the number of bits needed to encode a state and the locations of all p pebbles.

The game player learns which questions to ask the helper by simulating the JAG algorithm on the graph $G(\vec{\gamma}', ?, \vec{\beta}')$ starting in the configuration $\left\langle Q_{\langle l, \vec{\alpha}_l \rangle}, \Pi_{\langle l, \vec{\alpha}_l \rangle} \right\rangle$. The only thing preventing the game player from running the JAG is that he does not know the vector $\vec{\alpha}_l$. However, he can run the JAG as long as the computation does not depend on this unknown information. Specifically, suppose during the simulation that pebbles enter a fly swatter defined by a $\alpha_{\langle l,i \rangle}$ that is not known by the game player. The game player will be able to continue running the JAG for quite a while. However, as soon as the computation depends on which crossover edges of the fly swatter are switched, he must stop the simulation. He then asks the game helper a question about the game instance $\alpha_{\langle l,i \rangle}$. By definition of the parity game, the game helper reveals

to him the entire vector $\alpha_{\langle l,i\rangle}$. With this new information, the game player is able to continue the simulation until the next such event occurs.

As was done in section 4, let $S_0 \subset [1 \dots d]$ consist of those i for which some copy of $G(\vec{\gamma}', \vec{\alpha}_l)$ initially (i.e., according to $\Pi_{\langle l,\vec{\alpha}_l\rangle}$) contains a pebble in its ith fly swatter. Note that the p pebbles of the JAG might be contained in different copies of $G(\vec{\gamma}', \vec{\alpha}_l)$, but all such copies are considered. The game player begins the game by asking an arbitrary parity question E_i about $\alpha_{\langle l,i\rangle}$ for each $i \in S_0$.

The game player then starts simulating the JAG. Because he knows $\alpha_{\langle l,i\rangle}$ for every fly swatter containing pebbles, he can run the JAG at least until a pebble moves into an adjacent fly swatter. The JAG might alternately move pebbles contained in different copies of $G(\vec{\gamma}', \vec{\alpha}_l)$. However, we will count the number of time steps taken in each copy separately.

If the ith fly swatter in some copy of $G(\vec{\gamma}', \vec{\alpha}_l)$ is entered via both its s_i and t_i distinguished nodes, then the game player will also ask an arbitrary parity question E_i about $\alpha_{\langle l,i\rangle}$, as long as a question has not been asked about $\alpha_{\langle l,i\rangle}$ already. The indexes for which this happens forms the set S_1, as defined in section 4. By Claim 1, $|S_0 \cup S_1| \leq 2p+1$. Therefore, this completes at most $2p+1$ of the d game instances $\alpha_{\langle l,1\rangle}, \dots, \alpha_{\langle l,d\rangle}$. The remaining fly swatters indexed by $i \in [1 \dots d] - (S_0 \cup S_1)$ are *forward traversed*.

Two events will now be defined. If one of these events occurs within the ith fly swatter in some copy of $G(\vec{\gamma}', \vec{\alpha}_l)$, then the game player asks a question about the ith game instance $\alpha_{\langle l,i\rangle}$. Below, we prove that the computation of the JAG through the ith fly swatter does not depend on $\alpha_{\langle l,i\rangle}$ until one of these events occurs. It follows that the game player is always capable of running the simulation and hence of executing the parity-game protocol currently being defined. We also prove that, because fly swatters indexed by $i \in [1 \dots d] - (S_0 \cup S_1)$ are *forward traversed*, one of the events eventually occurs within each of them. From this, it follows that the parity-game protocol will terminate, asking a question about each game instance. Finally, I prove that traversing a fly swatter is at least twice as costly for the JAG when the question asked has even parity. Hence, the total cost for the parity game is proportional to the total cost for the JAG traversal.

Before continuing, recall that section 6 described two ways to ask a parity question. The first event will be defined so that it occurs within a fly swatter when the two pebble method is used within it, i.e., a pebble is left as a marker while another pebble walks along a sequence of super edges. Similarly, the second event will be defined so that it occurs when the one pebble method is used; i.e., a pebble walks up the handle, through a sequence of switches, and back down a handle again.

The first of these events is formally defined to occur within a fly swatter after the following has occurred twice during disjoint intervals of time. What must occur twice is that one of the super edges of the fly swatter is traversed, and during the entire time of its traversal, there is a pebble (not necessarily the same pebble at each time step) contained in the copy of $G(\vec{\gamma}', \vec{\alpha}_l)$ containing the fly swatter but not contained in the super edge in question. If this occurs in the ith fly swatter in some copy of $G(\vec{\gamma}', \vec{\alpha}_l)$, then the player asks an arbitrary question E_i about $\alpha_{\langle l,i\rangle}$.

The second event is defined to occur within the ith fly swatter. If it is initially empty, pebbles traverse up the handle, reach the cross-over edges, and then traverse down the handle again, reaching one of the distinguished nodes $s_{\langle l,i\rangle}$ or $t_{\langle l,i\rangle}$; yet during this entire time the first event never occurs within the ith fly swatter. We claim that for this event to occur, for $i \in [1 \dots d] - (S_0 \cup S_1)$, all the pebbles within

this fly swatter must traverse a single well-defined sequence of switches. When the second event occurs within the ith fly swatter in some copy of $G(\vec{\gamma}', \vec{\alpha}_l)$, the game player asks the question E_i that contains j iff the jth switch was traversed an odd number of times.

Now we will prove the claim. As stated, the pebbles are initially outside of the fly swatter. Because $i \notin S_1$, pebbles do not enter the fly swatter via both the distinguished nodes $s_{\langle l,i \rangle}$ and $t_{\langle l,i \rangle}$. Without loss generality assume that they enter via $s_{\langle l,i \rangle}$. Furthermore, there are never two pebbles within the fly swatter that have four or more full super edges between them. The reason is as follows. The pebbles contained in the fly swatter are initially together, because they enter only via $s_{\langle l,i \rangle}$. In order for two pebbles to get three full super edges between them, a super edge must be traversed while there is a pebble contained in this copy of $G(\vec{\gamma}', \vec{\alpha}_l)$ but not contained in the super edge in question. For the first event to occur, this must happen twice during disjoint intervals of time. For two pebbles to have four full super edges between, a second super edge must be traversed in this way. These two traversals occur during disjoint intervals in time and hence the first event occurs. Because the first event does not occur, no two pebbles in the fly swatter ever have four full super edges between them. Hence, the pebbles must traverse up the handle more or less together. They cannot traverse in opposite directions around a square of six super edges; hence they must traverse the same sequence of switches. Finally, they must traverse down the handle together. This proves the claim.

We will now prove that the game player always has enough information to continue running the JAG. Because of the game-helper's message and the fact that $\vec{\gamma}'$ and $\vec{\beta}'$ are fixed, the only information that the player is lacking is $\vec{\alpha}_l$. In addition, $\alpha_{\langle l,i \rangle}$ is revealed as soon he asks a question about it. Therefore, the only concern is whether the game player, even though he does not know $\alpha_{\langle l,i \rangle}$, can run the JAG as it traverses the ith fly swatter, at least until one of the two events happens. As said, if the first event has not occurred, then all the pebbles must traverse the same sequence of switches. Therefore, the only influence that $\alpha_{\langle l,i \rangle}$ (i.e., which switchable edges are switched) has on the computation is in which of the two fly swatter halves these pebbles are contained. However, the JAG has no way of knowing which half the pebbles are in, because the super edges in the two halves, the structure of the halves, and even the edge labels are identical. Therefore, the player knows as much as the JAG knows (i.e., the state and the partition of the pebbles) until second event occurs.

It has now been proven that the above protocol is well defined and meets the requirements of the game. The remaining task is to prove that the cost to the parity game is proportional to the cost of the JAG's traversal. Let us first consider the cost of the JAG's traversal. The JAG is run on the input graph $G(\vec{\gamma}', \vec{\alpha}_l, \vec{\beta}')$, starting in the configuration $\langle Q_{\langle l,\vec{\alpha}_l \rangle}, \Pi_{\langle l,\vec{\alpha}_l \rangle} \rangle$ specified by the JAG helper, until some copy of $G(\vec{\gamma}', \vec{\alpha}_l)$ (a line of fly swatters at the lth level) is traversed. By definition, the time of this traversal is $T\left[l, \langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \rangle, Q_{\langle l,\vec{\alpha}_l \rangle}, \Pi_{\langle l,\vec{\alpha}_l \rangle}\right]$, the number of pebbles "used" during this traversal is $p\left[l, \langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \rangle, Q_{\langle l,\vec{\alpha}_l \rangle}, \Pi_{\langle l,\vec{\alpha}_l \rangle}\right]$, and the cost incurred by the JAG is $w\left[l, \langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \rangle\right] = \min_{\langle Q,\Pi \rangle} h^{p[l, \langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \rangle, Q, \Pi]} T\left[l, \langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \rangle, Q, \Pi\right]$. Abbreviate these values to $T[l, \vec{\alpha}_l]$, $p[l, \vec{\alpha}_l]$ and $w[l, \vec{\alpha}_l]$. For each $i \in [1 \ldots d] - S_0$, define $T[l, \vec{\alpha}_l, i]$ to be the number of time steps for the event to occur within the ith fly swatter of some copy of $G(\vec{\gamma}', \vec{\alpha}_l)$. This time is at least the product of the number of super edges traversed and the minimum time to traverse a super edge on this input graph. The number of super edges traversed depends on whether a pebble was left as a marker within the fly swatter and whether a path from $s_{\langle l,i \rangle}$ to $t_{\langle l,i \rangle}$ was found in the first attempt, i.e.,

whether the answer to the parity was odd. The minimum cost to traverse a super edge $G(\vec{\gamma}')$ when the input graph is $G(\vec{\gamma}', \vec{\alpha}_l, \vec{\beta}')$ is defined to be $w\big[l-1, \big\langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \big\rangle\big]$. Each traversal of a super edge will cost at least this minimum. Abbreviate this with $w[l-1, \vec{\alpha}_l]$. The following claim bounds the time $T[l, \vec{\alpha}_l, i]$ for the event to occur within the ith fly swatter.

CLAIM 2. *For each* $i \in [1 \ldots d] - (S_0 \cup S_1)$,

$$T[l, \vec{\alpha}_l, i] \geq \left\{ \begin{array}{ll} 1 & \text{if } \bigoplus_{j \in E_i} \big[\alpha_{\langle l, i \rangle}\big]_j = 1 \\ 2 & \text{otherwise} \end{array} \right\} w[l-1, \vec{\alpha}_l] h^{-p\left[l, \vec{\alpha}_l\right]+1}.$$

Proof. Case 1. Suppose the first event occurs; i.e., two super edges at level $l-1$ are traversed by a pebble, and during the entire time of their traversal, there is another pebble contained in the same copy of $G(\vec{\gamma}', \vec{\alpha}_l)$ but not contained in these two super edges. Consider one of these two super edges traversed and let $\big\langle Q_{\langle l-1, \vec{\alpha}_l \rangle}, \Pi_{\langle l-1, \vec{\alpha}_l \rangle} \big\rangle$ be the configuration of the JAG at the beginning of this traversal. The number of time steps starting in this configuration until some copy of $G(\vec{\gamma}')$ is traversed (clearly the super edge in question) is defined to be $T\big[l-1, \big\langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \big\rangle, Q_{\langle l-1, \vec{\alpha}_l \rangle}, \Pi_{\langle l-1, \vec{\alpha}_l \rangle}\big]$ and the number of pebbles "used" in this traversal is defined to be $p\big[l-1, \big\langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \big\rangle, Q_{\langle l-1, \vec{\alpha}_l \rangle}, \Pi_{\langle l-1, \vec{\alpha}_l \rangle}\big]$. Abbreviate these to $T[l-1, \vec{\alpha}_l]$ and $p[l-1, \vec{\alpha}_l]$. It will also be useful to use $T'[l-1, \vec{\alpha}_l]$ to denote the time interval during which this super edge is traversed and to use $T'[l, \vec{\alpha}_l]$ to denote the time interval during which the entire line of fly swatters $G(\vec{\gamma}', \vec{\alpha}_l)$ is traversed. The "cost" of the traversal of the super edge is defined to be

$$h^{p[l-1, \vec{\alpha}_l]} T[l-1, \vec{\alpha}_l].$$

This is at least

$$w[l-1, \vec{\alpha}_l] = \min_{\langle Q, \Pi \rangle} h^{p\left[l-1, \left\langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \right\rangle, Q, \Pi\right]} \; T\big[l-1, \big\langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \big\rangle, Q, \Pi\big],$$

which is the minimal cost of traversing any super edge when the helper has preset the JAG configuration $\langle Q, \Pi \rangle$ to minimize the cost. Solving for the traversal time gives

$$T[l-1, \vec{\alpha}_l] \geq w[l-1, \vec{\alpha}_l] h^{-p\left[l-1, \vec{\alpha}_l\right]}.$$

The next step is to bound the number of pebbles $p[l-1, \vec{\alpha}_l]$ "used" to traverse this super edge. The intuition is as follows. The JAG has $p[l, \vec{\alpha}_l]$ pebbles available to traverse a copy of $G(\vec{\gamma}', \vec{\alpha}_l)$. If it leaves a pebble as a marker in the lth level and traverses a sequence of switchable edges with the other $p[l, \vec{\alpha}_l] - 1$ pebbles, then only $p[l, \vec{\alpha}_l] - 1$ pebbles are available for the traversal of these super edges $G(\vec{\gamma}')$. More formally, we want to prove that $p[l-1, \vec{\alpha}_l] \leq p[l, \vec{\alpha}_l] - 1$. To do this, we must bound the minimum number of copies of $G(\vec{\gamma}')$ in $G(\vec{\gamma}', \vec{\alpha}_l, \vec{\beta}')$ that contain pebbles (number of markers) during the traversal of this super edge. By definition,

$$p[l, \vec{\alpha}_l] \;=\; \max_{T \in T'[l, \vec{\alpha}_l]} \left[p + 1 - (\#\text{ copies of } G(\vec{\gamma}', \vec{\alpha}_l) \text{ containing pebbles at time } T)\right].$$

Therefore,

$$\min_{T \in T'[l-1, \vec{\alpha}_l]} [\#\text{ copies of } G(\vec{\gamma}', \vec{\alpha}_l) \text{ containing pebbles at time } T] \;\geq\; p - p[l, \vec{\alpha}_l] + 1.$$

We know that during the time interval $T'[l-1,\vec{\alpha}_l]$, one of the copies of $G(\vec{\gamma}',\vec{\alpha}_l)$ contains two copies of $G(\vec{\gamma}')$ (super edges) that contain pebbles. Therefore,

$$\min_{T\in T'[l-1,\vec{\alpha}_l]}[\#\text{ copies of } G(\vec{\gamma}') \text{ containing pebbles at time } T] \;\geq\; p - p[l,\vec{\alpha}_l] + 1 + \mathbf{1}$$

and, therefore, $p[l-1,\vec{\alpha}_l] \leq p[l,\vec{\alpha}_l] - 1$. From this we can bound the time of this super edge's traversal to be $T[l-1,\vec{\alpha}_l] \geq w[l-1,\vec{\alpha}_l]h^{-p[l,\vec{\alpha}_l]+1}$. Because the first event occurred, this occurred twice during disjoint intervals in time. Hence, the time required for the first event to occur can be taken to be at least the sum of the times for the two super edges to be traversed, without overcounting time steps. Therefore, $2 \times w[l-1,\vec{\alpha}_l]h^{-p[l,\vec{\alpha}_l]+1}$ time steps are required.

Case 2. Suppose the second event occurs and $\bigoplus_{j\in E_i}\left[\alpha_{\langle l,i\rangle}\right]_j = 1$. This event involves traversing up and down the handle. The handle contains $\frac{h}{2}$ super edges. Therefore, at least h super edges are traversed. These traversals must occur during disjoint intervals of time, because a super edge along the handle must be completely traversed before the next super edge along the handle is entered. The maximum number of pebbles $p[l-1,\vec{\alpha}_l]$ that can be used to traverse each of these copies of $G(\vec{\gamma}')$ is $p[l,\vec{\alpha}_l]$. Even if this number is used, the traversal time for each is $T[l-1,\vec{\alpha}_l] \geq w[l-1,\vec{\alpha}_l]h^{-p[l,\vec{\alpha}_l]}$. Therefore, the time to traverse h super edges is at least $w[l-1,\vec{\alpha}_l]h^{-p[l,\vec{\alpha}_l]+1}$.

Case 3. Suppose the second event occurs and $\bigoplus_{j\in E_i}\left[\alpha_{\langle l,i\rangle}\right]_j = 0$. Without loss of generality, assume that the pebbles entered the ith fly swatter from the $(i-1)$st fly swatter through the $s_{\langle l,i\rangle}$ distinguished node. The pebbles then traverse up the handle through a sequence of switches specified by E_i and back down the handle to a distinguished node. Because $\bigoplus_{j\in E_i}\left[\alpha_{\langle l,i\rangle}\right]_j = 0$, the pebbles do not make it to the distinguished node $t_{\langle l,i\rangle}$ but arrive back at the $s_{\langle l,i\rangle}$ node. How do we know that the pebbles traverse back up and down the handle a second time? By the definition of $i \notin S_1$, the JAG must "continue on" to traverse into the $i+1$st fly swatter. Hence, they must traverse up and down the handles a second time. The two traversals up and down the handle take at least $2 \times w[l-1,\vec{\alpha}_l]h^{p\left[l,\vec{\alpha}_l\right]+1}$ time steps. □

Using this claim, we can bound the total cost $w[l,\vec{\alpha}_l]$ (and time $T[l,\vec{\alpha}_l]$) for the JAG to traverse a line of fly swatters $G(\vec{\gamma}',\vec{\alpha}_l)$.

$$\begin{aligned} w[l,\vec{\alpha}_l] &= h^{p\left[l,\vec{\alpha}_l\right]}T[l,\vec{\alpha}_l] \\ &\geq h^{p\left[l,\vec{\alpha}_l\right]} \times \left[\sum_{i\in[1\ldots d]-(S_0\cup S_1)} \left\{ \begin{array}{ll} 1 & \text{if } \bigoplus\limits_{j\in E_i}[\alpha_i]_j = 1 \\ 2 & \text{otherwise} \end{array} \right\} w[l-1,\vec{\alpha}_l]h^{-p\left[l,\vec{\alpha}_l\right]+1}\right]. \end{aligned}$$

This can now be compared with the cost to the parity game defined above. By definition of the game, the number of questions asked per game instance on input $\vec{\alpha}_l$ is

$$\begin{aligned} c_{\vec{\alpha}_l} &= \frac{1}{d}\sum_{i\in[1\ldots d]} \left\{ \begin{array}{ll} 1 & \text{if } \bigoplus\limits_{j\in E_i}[\alpha_i]_j = 1 \\ 2 & \text{otherwise} \end{array} \right\} \\ &\leq \frac{1}{d}\left[2|S_0\cup S_1| \;+ \sum_{i\in[1\ldots d]-(S_0\cup S_1)} \left\{ \begin{array}{ll} 1 & \text{if } \bigoplus\limits_{j\in E_i}[\alpha_i]_j = 1 \\ 2 & \text{otherwise} \end{array} \right\}\right]. \end{aligned}$$

By Claim 1, $|S_0\cup S_1| \leq 2p+1$. This gives that

$$w[l,\vec{\alpha}_l] \geq w[l-1,\vec{\alpha}_l] \times h \times (dc_{\vec{\alpha}_l} - 4p - 2).$$

If $\vec{\alpha}_l$ were such that $w[l-1,\vec{\alpha}_l] \geq W[l-1]$ and $c_{\vec{\alpha}_l} \geq 1.5-\epsilon$, then

$$w[l,\vec{\alpha}_l] \geq W[l-1] \times h \times (d(1.5-\epsilon) - 4p - 2) \;=\; W[l].$$

It follows that

$$\Pr_{\vec{\alpha}_l}\left[w[l,\vec{\alpha}_l] < W[l]\right] \leq \Pr_{\vec{\alpha}_l}\left[c_{\vec{\alpha}_l} < 1.5-\epsilon\right] \Pr_{\vec{\alpha}_l}\left[w[l-1,\vec{\alpha}_l] < W[l-1]\right]. \qquad \square$$

11. Completing the proof. The final step is to prove the theorem.

THEOREM 1′ *For every constant $z \geq 2$ and every helper-JAG algorithm for st-traversal that uses $p \leq \frac{1}{29z}\frac{\log n}{\log\log n}$ pebbles and $q \leq 2^{\log^z n}$ states and whose traversal time on input $G(\vec{\alpha}_1,\ldots,\vec{\alpha}_L)$ is $T_{\langle\vec{\alpha}_1,\ldots,\vec{\alpha}_L\rangle}$,*

$$\Pr_{\langle\vec{\alpha}_1,\ldots,\vec{\alpha}_L\rangle}\left[T_{\langle\vec{\alpha}_1,\ldots,\vec{\alpha}_L\rangle} < n \times 2^{\frac{1}{29z}\frac{\log n}{\log\log n}}\right] \leq 2 \times 2^{-0.05\log^z n}.$$

Proof of Theorem 1′. Fix any constant $z \geq 2$ and consider a helper-JAG algorithm that uses $p \leq \frac{1}{29z}\frac{\log n}{\log\log n}$ pebbles and $q \leq 2^{\log^z n}$ states. The number of bits of help given by the helper JAG to set the initial configuration is the space of the JAG which is $b = p\log n + \log q \leq \frac{1}{29z}\frac{\log^2 n}{\log\log n} + \log^z n = (1+o(1))\log^z n$.

Consider the parity game with the bits of help from the helper being $b = (1+o(1))\log^z n$, the length of the vector $\vec{\alpha}_i$ being $r = 8$, the number of game instances being $d = 60\log^z n$, and the error tolerance being $\epsilon = 0.1$. By Lemma 1,

$$\begin{aligned}\Pr_{\vec{\alpha}}\left[c_{\vec{\alpha}} < 1.5-\epsilon\right] &\leq 2^b \times e^{-(2\epsilon^2 - 2^{-r+1})d} \\ &= 2^{(1+o(1))\log^z n} \times e^{-(2(.1)^2 - 2^{-8+1})60\log^z n} \\ &\leq 2^{-0.054\log^z n}.\end{aligned}$$

Define the line of fly swatter graphs such that the number of switches per fly swatter graph is $r = 8$, the length of the handle is $h = \frac{10r}{0.05} = 1600$, and number of fly swatters in the line is $d = 60\log^z n$. Then, the number of vertices in $G(\vec{\alpha}_1,\ldots,\vec{\alpha}_L)$ is $n \leq [(h+10r)\,d]^L = [1.05hd]^L$ and $L \geq \frac{\log n}{\log(1.05hd)} \geq \frac{\log n}{z\log\log n + O(1)}$. Corollary 1 bounds the time $T_{\langle\vec{\alpha}_1,\ldots,\vec{\alpha}_L\rangle}$ for the helper JAG to traverse input $G(\vec{\alpha}_1,\ldots,\vec{\alpha}_L)$:

$$\begin{aligned}&\Pr_{\langle\vec{\alpha}_1,\ldots,\vec{\alpha}_L\rangle}\left[T_{\langle\vec{\alpha}_1,\ldots,\vec{\alpha}_L\rangle} < [h\times(d(1.5-\epsilon)-4p-2)]^L\, h^{-p+1}\right] \\ &\leq L \times \Pr_{\vec{\alpha}}\left[c_{\vec{\alpha}} < 1.5-\epsilon\right] \\ &\leq \frac{\log n}{z\log\log n + O(1)} \times 2^{-0.054\log^z n} \;\leq 2\times 2^{-0.05\log^z n}.\end{aligned}$$

This bound on the time can be computed to be

$$\begin{aligned}&[h\times(d(1.5-\epsilon)-4p-2)]^L \times h^{-p+1} \\ &\geq \frac{n}{[1.05hd]^L} \times [(1.4-o(1))hd]^L \times h^{-p+1} \\ &\geq n \times \left[\frac{1.4-o(1)}{1.05}\right]^L \times h^{-p} \\ &\geq n \times 2^{0.415\frac{\log n}{z\log\log n + O(1)}} \times 2^{-\log(1600)\times\frac{1}{29z}\frac{\log n}{\log\log n}} \\ &\geq n \times 2^{\frac{1}{29z}\frac{\log n}{\log\log n}}. \qquad \square\end{aligned}$$

The proof of Theorem 1 follows from Theorem 1′ using Yao [25] and Lemma 2.

12. The pseudo-random walk game. The *st*-connectivity lower bound in Theorem 1 on the recursive fly swatter graphs reveals to the JAG every time in reaches the bottom of the handle whether it has reached the node $s_{\langle l,i\rangle}$ or the node $t_{\langle l,i\rangle}$. In reality, however, the JAG does not have access to this information. My belief in fact is that the JAG quickly looses track of where the pebble is located in the line of fly swatter graphs. In effect, I believe that the pebble performs a "pseudo-random walk" on the line. In this section, I define a game that characterizes this idea. A lower bound on this game would give the desired lower bound for *st*-connectivity of $n^{2-o(1)}$. The game is defined as follows.

GAME 1. *The parameters of the game are d, r, and $\tilde{W}$ (ideally r is a constant, but it could be as much as $\log d$, and ideally $\tilde{W} = d^{2-\epsilon}$, but it could be as little as $2d$). The game consists of a player and a randomly chosen input, both of which direct a pebble walking pseudo randomly on a line of length d. First, the player outputs a fixed sequence of vectors $\vec{\gamma} = \langle \gamma_1, \gamma_2, \ldots \rangle \in [\{0,1\}^r]^*$. The vector γ_t is used at time t. Then the input is chosen. It is a sequence of vectors $\vec{\alpha} = \langle \alpha_1, \alpha_2, \ldots, \alpha_d \rangle \in (\{0,1\}^r - \{0^r\})^d$ that is chosen uniformly at random. The vector α_i is associated with the ith node in the path.*

At every time step t, the pebble is located at some node i in the line and has a direction of travel from $\{\rightarrow, \leftarrow\}$. Initially, the pebble is at node 0 with direction $\rightarrow$. The transition depends on $\alpha_i \cdot \gamma_t = \bigoplus_{j \in [1\ldots r]} [\alpha_i]_j \wedge [\gamma_t]_j$. If this dot product is 1, then the pebble takes a step in the direction that it is traveling in. If the dot product is 0, then the pebble turns around before taking a step. The following table lists the possibilities.

	$\alpha_i \cdot \gamma_t = 1$	$\alpha_i \cdot \gamma_t = 0$
$\overrightarrow{i}$	$i \overrightarrow{+} 1$	$i \overleftarrow{-} 1$
$\overleftarrow{i}$	$i \overleftarrow{-} 1$	$i \overrightarrow{+} 1$

Consider a fixed player specification $\vec{\gamma}$. If the pebble ever returns to node 0, then the pebble stops and the player looses. Otherwise, let $W_{\vec{\alpha}}$ be the number of time steps for the pebble to reach node d when the input is $\vec{\alpha}$. The goal of the player is to move the pebble quickly from node 0 to node d. Our goal is to prove that no matter what the player does, for a random input, the pebble will almost always require at least $\tilde{W}$ step; i.e., we win if

$$\forall \vec{\gamma} \ \Pr_{\vec{\alpha}}(W_{\vec{\alpha}} < \tilde{W}) \leq 2^{-d^{\epsilon}}.$$

The fact that the player loses if the pebble ever returns to node 0 is not in itself significant. The game could equivalently be defined so that the pebble bounces at node 0 back to node 1 or that the line is extended in both directions. What is significant about this part of the definition is that the player does not interact at all with the game. He outputs the vectors $\vec{\gamma} = \langle \gamma_1, \gamma_2, \ldots \rangle$ at the beginning of the game without any information about the input $\vec{\alpha}$.

I think that a constant $r \in O(1)$ random bits per node is sufficient. In Theorem 1, it is 8. This is large enough so that $\{0,1\}^r - \{0^r\}$ is not significantly far from $\{0,1\}^r$. On the other hand, perhaps more random bits may help us. However, we have reasons to believe that having $r > \log d$ can only help the player.

LEMMA 4. *For $r \in O(1)$ and $\tilde{W} = d^{2-\epsilon}$, the player can win.*

Proof. For a purely random walk on a line, $\Pr_{\vec{\alpha}}(W_{\vec{\alpha}} < d^{2-\epsilon}) \leq 2^{-d^{\epsilon}}$. The player can easily be given to match this bound by outputting a random sequence of $\vec{\gamma}$. ∎

My conjecture is that the player in this random-walk game cannot do significantly better than by outputting a random sequence of $\vec{\gamma}$.

CONJECTURE 1. *For some $r \in [8 \ldots \log d]$ and for some $\tilde{W} \in [d^{1+\epsilon} \ldots d^{2-\epsilon}]$, the player must lose.*

Lemma 1 effectively proves this conjecture for $r = 8$ and $\tilde{W} = 2d$. The following theorem states the JAG result.

THEOREM 2. *Suppose that* Conjecture 1 *is true; i.e., the pebble in* Game 1 *almost always takes d^{ω} time steps to traverse a line of length d for some $\omega \in [1 + \epsilon \ldots 2 - \epsilon]$. It follows that for every constant $z > 2$, the expected time to solve undirected st-connectivity on a JAG with $p \leq \frac{\delta^2 \log n}{2z \log \log n}$ pebbles and $q \leq 2^{\log^z n}$ states is at least $n^{\omega(1-2\delta)}$.*

The proof reduces *st*-traversal of recursive fly swatter graphs by a helper JAG to the pseudo-random walk game. This reduction requires quite a different proof than the one presented here. It has not been included because it is slightly more difficult and because a lower bound on the pseudo-random walk game has not been obtained.

REFERENCES

[1] R. ALELIUNAS, R. M. KARP, R. J. LIPTON, L. LOVÁSZ, AND C. RACKOFF, *Random walks, universal traversal sequences, and the complexity of maze problems*, in Proc. 20th Annual Symposium on Foundations of Computer Science, IEEE, San Juan, Puerto Rico, October 1979, pp. 218–223.

[2] G. BARNES AND U. FEIGE, *Short random walks on graphs*, in Proc. 25th Annual ACM Symposium on Theory of Computing, San Diego, CA, May 1993, pp. 728–737.

[3] G. BARNES, J. F. BUSS, W. L. RUZZO, AND B. SCHIEBER, *A sublinear space, polynomial time algorithm for directed s-t connectivity*, in Proc., Structure in Complexity Theory, Seventh Annual Conference, IEEE, Boston, MA, June 1992, pp. 27–33; SIAM J. Comput., 27 (1998), pp. 1273–1282.

[4] G. BARNES AND J. EDMONDS, *Time-space lower bounds for directed st-connectivity on JAG models*, in Proc. 34th Annual Symposium on Foundations of Computer Science, Palo Alto, CA, November 1993, pp. 228–237.

[5] P. BEAME, A. BORODIN, P. RAGHAVAN, W. L. RUZZO, AND M. TOMPA, *Time-space tradeoffs for undirected graph traversal by graph automata*, Inform. Comput., 130 (1996), pp. 101–129.

[6] P. BERMAN AND J. SIMON, *Lower bounds on graph threading by probabilistic machines*, in Proc. 24th Annual Symposium on Foundations of Computer Science, IEEE, Tucson, AZ, November 1983, pp. 304–311.

[7] A. BORODIN, W. L. RUZZO, AND M. TOMPA, *Lower bounds on the length of universal traversal sequences*, J. Comput. System Sci., 45 (1992), pp. 180–203.

[8] A. BORODIN, *Structured vs. general models in computational complexity*, L'Enseignement Mathématique, 28 (1982).

[9] A. Z. BRODER, A. R. KARLIN, P. RAGHAVAN, AND E. UPFAL, *Trading space for time in undirected s-t connectivity*, in Proc. 21st Annual ACM Symposium on Theory of Computing, Seattle, WA, May 1989, pp. 543–549.

[10] A. K. CHANDRA, P. RAGHAVAN, W. L. RUZZO, R. SMOLENSKY, AND P. TIWARI, *The electrical resistance of a graph captures its commute and cover times*, in Proc. 21st Annual ACM Symposium on Theory of Computing, Seattle, WA, May 1989, pp. 574–586.

[11] S. A. COOK AND C. W. RACKOFF, *Space lower bounds for maze threadability on restricted machines*, SIAM J. Comput., 9 (1980), pp. 636–652.

[12] J. EDMONDS, *Time-Space Lower Bounds for Undirected and Directed ST-Connectivity on JAG Models*, Ph.D. thesis, Department of Computer Science, University of Toronto, August 1993.

[13] J. EDMONDS, *Time-space trade-offs for undirected st-connectivity on a JAG*, in Proc. 25th Annual ACM Symposium on Theory of Computing, San Diego, CA, May 1993, pp. 718–727.

[14] R. Impagliazzo and D. Zuckerman, *How to recycle random bits*, in Proc. 30th Annual Symposium on Foundations of Computer Science, IEEE, Research Triangle Park, NC, October 1989, pp. 248–253.

[15] J. D. Kahn, N. Linial, N. Nisan, and M. E. Saks, *On the cover time of random walks on graphs*, J. Theoret. Probab., 2 (1989), pp. 121–128.

[16] *Logic and Algorithmic*, an International Symposium Held in Honor of Ernst Specker, Zürich, February 5–11, 1980. Monographie 30 de L'Enseignement Mathématique, Université de Genève, 1982.

[17] N. Nisan, E. Szemerédi, and A. Wigderson, *Undirected connectivity in $O(\log^{1.5} n)$ space*, in Proc. 33rd Annual Symposium on Foundations of Computer Science, IEEE, Pittsburgh, PA, October 1992, pp. 24–29.

[18] C. K. Poon, *A Sublinear Space, Polynomial Time Algorithm for Directed ST-Connectivity on the JAG Model*, Ph.D. thesis, University of Toronto, 1995.

[19] S. Rudich, personal communication, 1994.

[20] W. J. Savitch, *Relationships between nondeterministic and deterministic tape complexities*, J. Comput. System Sci., 4 (1970), pp. 177–192.

[21] W. J. Savitch, *Maze recognizing automata and nondeterministic tape complexity*, J. Comput. System Sci., 7 (1973), pp. 389–403.

[22] N. A. Spencer, John Wiley and Sons, Inc., 1992.

[23] R. E. Tarjan, *Depth-first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.

[24] A. Wigderson, *The complexity of graph connectivity*, in Mathematical Foundations of Computer Science 1992: Proceedings, 17th Symposium, I. M. Havel and V. Koubek, eds., Lecture Notes in Computer Science 629, Springer-Verlag, Prague, Czechoslovakia, August 1992, pp. 112–132.

[25] A. C. Yao, *Probabilistic computations: Toward a unified measure of complexity*, in Proc. 18th Annual Symposium on Foundations of Computer Science, IEEE, Providence, RI, October 1977, pp. 222–227.

SIAM J. COMPUT.
Vol. 27, No. 6, pp. 1515–1530, December 1998
© 1998 Society for Industrial and Applied Mathematics

ON LEARNING READ-k-SATISFY-j DNF *

HOWARD AIZENSTEIN†, AVRIM BLUM‡, RONI KHARDON§, EYAL KUSHILEVITZ¶, LEONARD PITT‖, AND DAN ROTH**

Abstract. We study the learnability of read-k-satisfy-j (RkSj) DNF formulas. These are boolean formulas in disjunctive normal form (DNF), in which the maximum number of occurrences of a variable is bounded by k, and the number of terms satisfied by any assignment is at most j. After motivating the investigation of this class of DNF formulas, we present an algorithm that for any unknown RkSj DNF formula to be learned, with high probability finds a logically equivalent DNF formula using the well-studied protocol of equivalence and membership queries. The algorithm runs in polynomial time for $k \cdot j = O(\frac{\log n}{\log \log n})$, where n is the number of input variables.

Key words. DNF, learning, computational learning theory, decision trees

AMS subject classifications. 68T05, 68Q25

PII. S0097539794274398

1. Introduction.

A central question in the theory of learning is to decide which subclasses of boolean formulas can be learned in polynomial time with respect to any of a number of reasonable learning protocols. Among natural classes of formulas, those efficiently representable in disjunctive normal form (DNF) have been extensively studied since the seminal paper of Valiant [Val84]. Nonetheless, whether for such formulas there are learning algorithms that can be guaranteed to succeed in polynomial time using any of the standard learning protocols remains a challenging open question.

Consequently, recent work has focused on the learnability of various restricted subclasses of DNF formulas. For example, the learnability of those DNF formulas with a bounded number of terms, or with bounded size terms, or with each variable appearing only a bounded number of times, has received considerable attention (see, e.g., [Val84, Ang87, PV88, BS90, Han91, Aiz93, PR95]). Investigation along these lines is of interest for two reasons. First, techniques developed for learning subclasses

*Received by the editors September 16, 1994; accepted for publication (in revised form) August 15, 1996; published electronically June 3, 1998.
http://www.siam.org/journals/sicomp/27-6/27439.html

† School of Medicine, University of Pittsburgh Western Psychiatric Institute and Clinic, 3811 O'Hara St., Pittsburgh, PA 15213 (aizen+@pitt.edu). This research was supported in part by NSF grant IRI-9014840.

‡ School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3891 (avrim@cs.cmu.edu). This research was supported in part by NSF National Young Investigator grant CCR-93-57793 and a Sloan Foundation Research Fellowship.

§ Aiken Computation Laboratory, Harvard University, Cambridge, MA 02138 (roni@das.harvard.edu). This research was supported by grant DAAL03-92-G-0115 (Center for Intelligent Control Systems).

¶ Department of Computer Science, Technion, Haifa, Israel 32000 (eyalk@cs.technion.ac.il, http://www.cs.technion.ac.il/∼eyalk). Part of this research was done while the author was at Aiken Computation Laboratory, Harvard University and was supported by research contracts ONR-N0001491-J-1981 and NSF-CCR-90-07677.

‖ Department of Computer Science, University of Illinois, Urbana, IL 61801 (pitt@cs.uiuc.edu). This research was supported in part by NSF grant IRI-9014840.

** Department of Computer Science, University of Illinois, Urbana, IL 61801 (danr@cs.uiuc.edu). Part of this work was done while the author was at Harvard University and was supported by NSF grant CCR-92-00884 and by DARPA grant AFOSR-F4962-92-J-0466.

of DNF may well be applicable to solving the more general problem, just as difficulties encountered may suggest approaches towards proving that the general problem admits no tractable solution. Second, and at least as important from a practical perspective, is that in many real-world machine-learning settings, the full generality of DNF may not be required, and the function to be learned might have an efficient expression in one of the restricted forms for which learning algorithms have been found.

In this paper we show (Theorem 19) that boolean formulas over n variables that can be expressed as *read-k-satisfy-j DNF* (which we abbreviate "RkSj") are efficiently learnable, provided that $k \cdot j = O(\frac{\log n}{\log \log n})$. An RkSj DNF representation for a function is a DNF formula in which the maximum number of occurrences of each variable is bounded by k and each assignment satisfies at most j terms of the formula. The special case of RkSj for $j = 1$ is referred to as *disjoint* DNF, and we denote it as RkD. The motivation for considering disjoint DNF expressions arises from the fact that they generalize the decision-tree representation of boolean functions, which have been used extensively in machine-learning theory and practice [BFOS84, Qui86, Qui93]. The "satisfy-j" constraint relaxes the notion of disjointness, thus incorporating a wider class of functions. In the next section we will discuss the relationships between RkSj DNF, RkD DNF, and previously investigated subclasses of DNF formulas.

The learning model we use is the standard equivalence and membership queries model [Ang88]: we assume that some unknown *target* RkSj formula F to be learned is chosen by nature. A learning algorithm may propose as a hypothesis any DNF formula H by making an *equivalence query* to an oracle. (This has sometimes been referred to as an "extended" equivalence query, as the hypothesis of the algorithm need not be in RkSj form.) If H is logically equivalent to F then the answer to the query is "yes" and the learning algorithm has succeeded and halts. Otherwise, the answer to the equivalence query is "no" and the algorithm receives a *counterexample*—a truth assignment to the variables that satisfies F but does not satisfy H (a *positive* counterexample) or vice-versa (a *negative* counterexample). The learning algorithm may also query an oracle for the value of the function F on a particular assignment (example) a by making a *membership query* on a. The response to such a query is "yes" if a is a positive example of F ($F(a) = 1$), or "no" if a is a negative example of F ($F(a) = 0$).

We say that the learner *learns* a class of functions $\mathcal{F}$ if for every function $F \in \mathcal{F}$ and for any confidence parameter $\delta > 0$, with probability at least $1-\delta$ over the random choices of the learner, the learner outputs a hypothesis H that is logically equivalent to F and does so in time polynomial in n (the number of boolean variables of F), $|F|$ (the length of the shortest representation of F in some natural encoding scheme), and $\frac{1}{\delta}$. This protocol is a probabilistic generalization of the slightly more demanding exact learning model, which requires that deterministically, in time polynomial in n and $|F|$, the learning algorithm necessarily finds some logically equivalent H. Learnability in this model implies learnability in the well-studied "PAC" model with membership queries [Val84, Ang88]. In that model, instead of equivalence queries, examples from an arbitrary, unknown probability distribution $\mathcal{D}$ are available, and the goal of the learner is to find with probability at least $1 - \delta$ a hypothesis H that disagrees with F on a set of assignments of probability at most ϵ measured according to $\mathcal{D}$.

The rest of the paper is organized as follows: we review related work in section 2. Section 3 includes preliminary definitions. In section 4 we prove some useful properties of RkSj DNF formulas. In section 5 we present the learning algorithm and prove its correctness. In section 6 we show that read-twice disjoint (R2D) DNF does not have

small conjunctive normal form (CNF) representation (and in particular, no small decision trees), which demonstrates that the class of R2D DNF (and the more general class RkSj) is incomparable to subclasses of DNF recently shown to be learnable.

2. Related results. When membership queries are not available to the learning algorithm, the problem of learning DNF, or even restricted subclasses of DNF, appears quite difficult. In the model of learning with equivalence queries only, or in the PAC model without membership queries [Val84], learning algorithms are known only for k-DNF formulas (DNF formulas with a constant number of literals in each term) [Val84, Ang88], DNF formulas with a constant number of terms [KLPV87, PV88] (provided that the hypotheses need not be in the same form), or for "polynomially explainable" subclasses of DNF in which the number of possible terms causing an example to be positive is bounded by a polynomial [Val85, KR95]. Using information-theoretic arguments, Angluin [Ang90] has shown that DNF in general is not learnable in polynomial time using just DNF equivalence queries, although the negative result does not extend to the PAC model.

In contrast, there are a number of positive learning results when membership queries are allowed, and in many of these cases it can be shown that without the membership queries the learning problem is as hard as the problem of learning general DNF. Among these learnable subclasses are: monotone DNF [Val84, Ang88], read-twice DNF [Han91, AP91, PR95], "Horn" DNF (at most one literal negated in every term) [AFP92], $\log n$ term DNF [BR92], and DNF $\cap$ CNF (this class includes decision trees) [Bsh95].

Read-k DNF. Particularly relevant to our work is the considerable attention that has been given to the learnability of boolean formulas where each variable occurs some bounded (often constant) number k of times ("read k"). Recently, polynomial-time algorithms have been given for learning read-once and read-twice DNF [AHK93, Han91, AP91, PR95]. (The result of [AHK93] is actually much stronger, giving an algorithm for learning any read-once boolean formula, not just those efficiently representable as read-once DNF.)

The learnability of read-k DNF for $k \geq 3$ seems to be much more difficult. Recent results show that in the PAC model with membership queries, learning read-thrice DNF is no easier than learning general DNF formulas [AK95, Han91]. In the PAC model, assuming the existence of one-way functions, Angluin and Kharitonov [AK95] have shown that membership queries cannot help in learning DNF (assuming that the learning algorithm is not required to express its hypotheses in any particular form, and that the distribution on examples is "polynomially bounded"). In other words, assuming one-way functions exist, in the PAC model the problem of learning read-thrice DNF with membership queries is no easier than learning general DNF without queries (in polynomially bounded distributions). In the model of learning from equivalence and membership queries, it has been shown ([PR94]; see also [AHP92]) that there is no polynomial-time algorithm for exactly learning read-thrice DNF using read-thrice DNF equivalence queries, unless P = NP. One way to view our results is to note that while the work referenced above indicates that for $k \geq 3$, learning read-k DNF is difficult, by adding a disjointness condition on the terms of the formula (or even a much weaker condition that each assignment satisfy only j terms) the learning problem becomes tractable.

Disjoint DNF. Variations of disjoint DNF formulas (without any "read" restriction) have also been considered, as disjoint DNF formulas are a natural generalization of decision trees. While every boolean function clearly has a disjoint DNF expression

(simply take a fundamental product of exactly n literals for each satisfying assignment), a function with a short DNF representation might have an exponentially long disjoint representation. However, every decision tree can be efficiently expressed as a disjoint DNF formula, by creating a term corresponding to the assignment of variables along each branch that leads to a leaf labeled "1". Recently, Bshouty [Bsh95] has given an algorithm (using equivalence and membership queries) for learning boolean formulas in time polynomial in the size of their DNF and CNF representations. Since every decision tree has a DNF and CNF that have size polynomial in the size of the original tree, his algorithm may be used to learn decision trees. We show (Theorem 20) that even R2D DNF formulas do not necessarily have small CNF representations (and therefore do not have small decision trees). In particular, we present a family of functions $\{F_n\}$ that have short (poly(n)) R2D DNF formulas but require CNF formulas of size $2^{\Omega(\sqrt{n})}$.

Other results concerning the learnability of disjoint DNF have also been obtained. On the positive side, Jackson [Jac94] gives a polynomial-time algorithm for learning arbitrary DNF formulas in the PAC model with membership queries, provided that the error is measured with respect to the uniform distribution. This extends results on learning decision trees, disjoint DNF, and satisfy-j DNF formulas in the same model [KM93, Kha94, BFJ$^+$94]. On the negative side, some recent hardness results for learning DNF in restricted models apply for disjoint DNF as well: Blum et al. [BFJ$^+$94] prove a hardness result for learning $\log n$ disjoint DNF in the "statistical queries" model (note that in this model the learner does not have access to a membership oracle). Aizenstein and Pitt [AP95] show that even if subset queries are allowed, disjoint DNF (and also read-twice DNF) is not learnable by algorithms that collect prime implicants in a greedy manner. Yet, the learnability for disjoint DNF remains unresolved in any reasonable learning model, other than the result above assuming a uniform distribution. Another way, then, to view our results, is to note that if we add a read-k restriction on disjoint DNF, we obtain a positive learning result.

Finally, note that RkSj DNF may be thought of as a generalization of k-term DNF (those DNFs with at most k terms): every k-term DNF formula is trivially an RkSj DNF formula. Thus, our results may also be viewed as an extension of previous results for learning k-term DNF formulas [Ang87, BR92, Bsh95], although a true generalization of the strongest of these results would learn RkSj DNF for $k + j = O(\log n)$. We leave the latter as an interesting open problem.

3. Preliminaries. Let $x_1, \ldots, x_n$ be n boolean variables. A *literal* is either a variable x_i or its negation $\overline{x_i}$. A literal ℓ is said to have *index* i if $\ell \in \{x_i, \overline{x_i}\}$. A *term* is a conjunction (AND) of literals; a DNF formula is a disjunction (OR) of terms. For convenience, we will also treat a set of literals as the term that consists of the conjunction of literals in the set, and we will treat a set of terms as the DNF formula obtained by taking the disjunction of terms in the set. An *example* (or *assignment*) a is a boolean vector over $\{0,1\}^n$, and we write $a[i]$ to denote the ith bit of a. An example a satisfies a term t (denoted by $t(a) = 1$) if and only if it satisfies all the literals in t. An example a satisfies a DNF formula F (written $F(a) = 1$) if and only if there is at least one term $t \in F$ that a satisfies. If a satisfies F then a is said to be a *positive example* of F; otherwise it is a *negative example*. An RkSj DNF formula F is a DNF formula in which each variable appears at most k times, and for every assignment a, there are at most j terms of F satisfied by a.

For two boolean functions F_1, F_2 we write that F_1 implies F_2 to mean that for every assignment a, if $F_1(a) = 1$ then $F_2(a) = 1$. A term t is an *implicant* of a DNF

formula F if t implies F; it is a *prime implicant* if no proper subset of t is also an implicant of F.

We often want to look at examples obtained by changing the value of a given example on a specified literal. Toward this end, we introduce the following notation. Let $a \in \{0,1\}^n$ be an example, and i an index. We define the example $b = \mathit{flip}(a,i)$ by: $b[i] = 1 - a[i]$ and $b[i'] = a[i']$ for $i' \neq i$. For instance, if $a = 1101$ then $\mathit{flip}(a,3) = 1111$. If ℓ is a literal of index i, then $\mathit{flip}(a,\ell)$ is defined to be the example $\mathit{flip}(a,i)$. Hence, if $a = 1101$, then $\mathit{flip}(a,x_3) = 1111$ and also $\mathit{flip}(a,\overline{x_3}) = 1111$. Usually, we will write $\mathit{flip}(a,\ell)$ only when assignment a satisfies literal ℓ. Finally, we may apply the flip operator to sets of literals as well, to obtain an assignment given by simultaneously flipping each literal in the set: for any set $I \subseteq \{1,\ldots,n\}$ define $\mathit{flip}(a,I)$ to be the assignment b such that $b[i] = 1 - a[i]$ if $i \in I$, and $b[i'] = a[i']$ for $i' \notin I$.

4. Properties of RkSj DNF formulas. A natural way to gather information about a DNF formula F by asking membership queries is to find assignments a such that when the assignment to some single literal of a is changed, the value of F changes. That is, for some literal ℓ, $F(\mathit{flip}(a,\ell)) \neq F(a)$. Such an assignment is called a *sensitive* assignment, and all such literals ℓ are sensitive literals for a. More formally, we present the following definition.

DEFINITION 1. *For an assignment $a \in \{0,1\}^n$ and a function F, the set of* sensitive *literals is given by* $\mathit{sensitive}(a) = \{\ell \mid F(\mathit{flip}(a,\ell)) \neq F(a)\}$.

Observe that the sensitive set of an assignment can be found by asking $n+1$ membership queries. With one membership query we find the value of $F(a)$, and then with n additional membership queries, we determine whether the value of $F(\mathit{flip}(a,\ell))$ differs from $F(a)$ for each of the n literals satisfied by a.

PROPERTY 2. *If F is any DNF formula and a is an assignment such that $F(a) = 1$, then $\mathit{sensitive}(a) \subseteq \cap\{t \in F : t(a) = 1\}$. In other words, each literal in $\mathit{sensitive}(a)$ appears in all terms satisfied by a.*

Proof. Suppose to the contrary that for some literal ℓ in $\mathit{sensitive}(a)$ and for some term t satisfied by a, ℓ is not in t. Then t would also be satisfied by $\mathit{flip}(a,\ell)$, which implies that $F(\mathit{flip}(a,\ell)) = 1$, contradicting the assumption that $\ell \in \mathit{sensitive}(a)$. □

Property 2 suggests that one way to find (at least part of) a term of a DNF formula, given a satisfying assignment a, is to construct the sensitive set of a. In the event that the sensitive set is *exactly* a term t satisfied by a, then we have made significant progress and can continue to find other terms. However, Property 2 only guarantees that each sensitive literal is in every term a satisfies; it does not guarantee that *every* literal of *some* term is in the sensitive set. Below, we will show that for RkSj DNF formulas, we can find assignments whose sensitive sets are not missing "many" literals. From these assignments, we will be able to find terms of the formula.

Suppose a satisfies only a single term t. By Property 2, any literal ℓ that is not in t will not be in $\mathit{sensitive}(a)$. But why might some literal ℓ that *is* in t fail to be in $\mathit{sensitive}(a)$? It must be because $\mathit{flip}(a,\ell)$ satisfies some term t' containing $\overline{\ell}$. Thus, t' was "almost" satisfied by assignment a; only the literal ℓ was set wrong. More formally, we have Definition 3.

DEFINITION 3. *A term t is* almost satisfied *by an assignment a with respect to index i if $t(a) = 0$ but $t(\mathit{flip}(a,i)) = 1$. A term t is* almost satisfied *by an assignment a with respect to (literal) ℓ if a satisfies literal ℓ, and if $t(a) = 0$ but $t(\mathit{flip}(a,\ell)) = 1$.*

Let $Y(a)$ be the set of all i's such that some term t is almost satisfied by a with respect to i. The following lemma is central in the analysis of our algorithm; it gives

a bound on the size of $Y(a)$. In the appendix it is shown that this bound is essentially optimal.

LEMMA 4. *Let F be an RkSj formula, and let $a \in \{0,1\}^n$. Then, $|Y(a)| \leq 2kj$.*

Proof. Let $a \in \{0,1\}^n$ be fixed. Note that for each almost satisfied term t, there is exactly one index i such that t is almost satisfied (by a) with respect to i. We partition the set of almost satisfied terms into $m = |Y(a)|$ nonempty equivalence classes $S_{i_1}, S_{i_2}, \ldots, S_{i_m}$ such that t is in S_i if and only if t is almost satisfied by a with respect to i. Suppose t is in S_i. If $a[i] = 0$ then t must contain the literal x_i, since flipping the ith bit from 0 to 1 causes t to become satisfied. Similarly, if $a[i] = 1$, then t must contain the literal $\overline{x_i}$. For notational convenience, define $S_i(x_i) = x_i$ if $a[i] = 0$, and $S_i(x_i) = \overline{x_i}$ if $a[i] = 1$. Thus, every term t in S_i contains the literal $S_i(x_i)$. Further, since F is read-k, for each equivalence class S_i, $|S_i|$ is at most k.

Consider a directed graph $G = (V, E)$ induced by the equivalence classes, with $V = \{S_{i_1}, S_{i_2}, \ldots, S_{i_m}\}$ and $E = \{\langle S_{i'}, S_i\rangle$: some term $t \in S_i$ contains the literal $\overline{S_{i'}(x_{i'})}\}$. (Note that $t \in S_i$ cannot contain the literal $S_{i'}(x_{i'})$ since this literal is negated in a, and t is almost satisfied by a with respect to a different literal.)

Since each of the (at least one) terms in S_i contains the literal $S_i(x_i)$, and since F is read-k, there are at most $k-1$ other equivalence classes $S_{i'}$ containing a term which has the literal $\overline{S_i(x_i)}$. Thus, the outdegree of every vertex is at most $k-1$. Suppose G' is any subgraph of G containing μ vertices. Since there are at most $\mu(k-1)$ edges in G', there must be some vertex $S \in G'$ whose indegree + outdegree in G' is at most $2\mu(k-1)/\mu = 2(k-1)$. That is, S has at most $2(k-1)$ neighbors in the underlying undirected graph of G'.

Now assume, contrary to this lemma, that $|Y(a)|$ ($= m$, the number of vertices in G), is larger than $2kj$. Then there must be an independent set $V_{ind} \subseteq V$ of G of size at least $j+1$: such an independent set V_{ind} can be constructed by first placing into V_{ind} some vertex S_i satisfying the indegree + outdegree bound above, and removing from G each of the at most $2(k-1)$ vertices adjacent to S_i in the underlying undirected graph. Then another vertex $S_{i'}$ is included in V_{ind}, its neighbors are removed, etc. After iteratively choosing j such vertices (no two of which are adjacent in the underlying undirected graph) and removing their neighbors, we will have eliminated at most $j \cdot (2(k-1)+1) = 2kj - j < 2kj$ vertices. Thus, we can include at least one more vertex into the set V_{ind}, so that $|V_{ind}| \geq j+1$ as desired.

Let $I = \{i : S_i \in V_{ind}\}$, namely, the set of indices of the at least $j+1$ equivalence classes with no edges between them in G. Consider a term t in some equivalence class S_i with $i \in I$. By definition of the equivalence classes, t is almost satisfied by a with respect to i, and thus $\mathit{flip}(a,i)$ will satisfy term t. Since no other literal in t has index in I by definition of the set V_{ind}, $\mathit{flip}(a,I)$ satisfies t as well. Thus $\mathit{flip}(a,I)$ satisfies every term from each of the $j+1$ equivalence classes indexed by I, contradicting the assumption that F was a satisfy-j formula. □

DEFINITION 5. *A term $\tilde{t}$ is a p-variant of a term t if it contains all the literals of t and at most p additional literals.*

COROLLARY 6. *If a is a satisfying assignment for an RkSj DNF formula F satisfying the single term $t \in F$, then t is a $2kj$-variant of sensitive(a).*

Proof. First observe by Property 2 that *sensitive*$(a) \subseteq t$. Thus to prove this corollary it is sufficient to show that there are at most $2kj$ literals in $t -$ *sensitive*(a). Consider why a literal ℓ with index i might be in t but not in *sensitive*(a). By the definition of sensitive set this means that $F(\mathit{flip}(a,i)) = F(a)$ and in this case $F(\mathit{flip}(a,i)) = F(a) = 1$. But ℓ in t implies that $t(\mathit{flip}(a,i)) = 0$, so $\mathit{flip}(a,i)$ must

satisfy some term other than t. Since t is the only term satisfied by a, all the terms satisfied by $\mathit{flip}(a, i)$ must be almost satisfied by a with respect to i. This means $i \in Y(a)$, and by Lemma 4 there can be at most $2kj$ such values of i, hence at most $2kj$ corresponding literals ℓ in $t - \mathit{sensitive}(a)$. □

DEFINITION 7. *Let $a, b \in \{0,1\}^n$. Then b is a* d-neighbor *of a if $b = \mathit{flip}(a, I)$ for some set I of cardinality at most d. (That is, if the Hamming distance between a and b is at most d.)*

LEMMA 8. *If a is a satisfying assignment for an RkSj DNF formula F satisfying exactly the terms $T = \{t_1, t_2, \ldots, t_q\} \subseteq F$ then there exists a $(j-1)$-neighbor b of a such that some term in T is a $2kj$-variant of $\mathit{sensitive}(b)$.*

Proof. Since F is satisfy-j and a is a satisfying assignment it must be the case that $1 \leq |T| = q \leq j$. Note by Corollary 6 that if $q = 1$, t_1 is a $2kj$-variant of $\mathit{sensitive}(a)$, and since a is a 0-neighbor of itself, the lemma follows. To see that the lemma is true for $1 < q \leq j$ we will show that in this case either some term satisfied by a is a $2kj$-variant of $\mathit{sensitive}(a)$ or else there exists some i such that $\mathit{flip}(a, i)$ satisfies a proper subset of T, and $F(\mathit{flip}(a, i)) = 1$. It then follows by induction that for some set of indices $I \subseteq \{1, \ldots, n\}$ with $|I| \leq j - 1$, some term satisfied by a is a $2kj$-variant of $\mathit{sensitive}(\mathit{flip}(a, I))$. The assignment $\mathit{flip}(a, I)$ is the $(j-1)$-neighbor b of a mentioned in the lemma.

If some term in T is a $2kj$-variant of $\mathit{sensitive}(a)$ then we are done. Otherwise each term in T must contain more than $2kj$ literals not in $\mathit{sensitive}(a)$. Let t be a term in T and let $L = \{\ell_1, \ell_2, \ldots\}$ be a set of more than $2kj$ literals in t that are not in $\mathit{sensitive}(a)$. By Lemma 4, $|Y(a)| \leq 2kj$, so there must be some literal $\ell \in L$ (say, with index i) such that there does not exist a term in F almost satisfied by a with respect to i. Consequently, $\mathit{flip}(a, i)$ cannot satisfy any term not satisfied by a, and must therefore satisfy a proper subset of the terms satisfied by a since t will no longer be satisfied. Since $\ell \notin \mathit{sensitive}(a)$, $\mathit{flip}(a, i)$ still satisfies at least one term. □

While the statement of Lemma 8 may appear somewhat confusing at first glance, if examined further we see that it immediately suggests an algorithm for learning RkSj DNF formulas (deterministically) where k and j are constants. First, conjecture the null DNF formula. On receiving a positive counterexample a, generate a collection H of terms, at least one of which is a term of the target DNF formula. To build the collection H, simply enumerate each $(j-1)$-neighbor b of a, and then include in H every $2kj$-variant of the sensitive set of each such b. By Lemma 8, H will necessarily contain some term of F that a satisfied. Of course, H may also contain many terms that are not in F (though only polynomially many, assuming k and j are constants). On seeing a negative counterexample, the algorithm simply removes from H any term that is satisfied, since these cannot be in F. Thus, on positive counterexamples, the algorithm adds to its hypothesis H at least one correct term not yet included, and at most a polynomial number of incorrect terms. On each negative counterexample, at least one of the incorrect terms is removed. It follows that the number of counterexamples that can be received (and hence the number of equivalence queries until the algorithm has produced a DNF logically equivalent to the target) is bounded by a polynomial in n and the number of terms of the target RkSj DNF formula.

This general approach of finding many possible terms from a positive counterexample a, and discarding the bad terms when seeing negative counterexamples, is a fairly standard approach used in many of the algorithms for learning subclasses of DNF formulas discussed in earlier sections. In the particular case of RkSj DNF, we

need to develop additional technical lemmas in order to devise an algorithm that works for values of k and j that are not constant.

The following is an immediate corollary of Lemma 4.

COROLLARY 9. *For all a such that $F(a) = 0$, at most $2kj$ of the neighbors of a (in the cube $\{0,1\}^n$) are satisfying assignments of F.*

Proof. The neighbors of a are exactly $\mathit{flip}(a,1), \ldots, \mathit{flip}(a,n)$. Hence if $\mathit{flip}(a,i)$ is a satisfying assignment then $i \in Y(a)$. □

Let $\mathcal{U}$ denote the uniform distribution, and let "$x \in \mathcal{U}$" denote that x is selected uniformly at random (in $\{0,1\}^n$). Also, let "true" be the "all 1's" concept.

LEMMA 10. *If F is an RkSj DNF, and $F \not\equiv$ true, then $\mathbf{Prob}_{x \in \mathcal{U}}[F(x) = 0] \geq 2^{-2kj}$*

Proof. The proof uses an argument similar to one in [Sim83]. Think of the cube $\{0,1\}^n$ as a graph in which each vertex is represented by an n-bit string and there is an edge between any two vertices of Hamming distance 1. Consider the vertex-induced subgraph $G = (V, E)$, where V is the set of points $a \in \{0,1\}^n$ such that $F(a) = 0$. Since $F \not\equiv$ true, we know that there is at least one vertex in V. We further know by Corollary 9 that the minimum degree of any vertex in G is at least $n - 2kj$.

A standard argument shows that any nonempty subgraph of $\{0,1\}^n$ with minimum degree at least d has at least 2^d vertices. For completeness we give the proof, which is by induction on n. The cases where $n = 1$ or $d = 0$ are easily verified. For the more general cases, notice that cutting the cube in any coordinate i leaves the two subcubes of dimension $n-1$, each with minimum degree at least $d-1$. This is because each vertex has at most one neighbor in the other subcube. So we simply choose i so that we have a nonempty subset of V in each subcube. Using the induction, we get at least $2 \cdot 2^{d-1} = 2^d$ vertices.

This implies that $|V| \geq 2^{n-2kj}$, and therefore $\mathbf{Prob}_{x \in \mathcal{U}}[F(x) = 0] \geq 2^{-2kj}$. □

LEMMA 11. *Let F be an RkSj formula. Then F has at most kjq terms of length (number of literals) at most q.*

Proof. Consider a graph with one vertex for each term in F of length at most q and an edge between two vertices if their corresponding terms are not simultaneously satisfiable. In other words, two terms are connected exactly when one of the terms contains some variable x_i and the other contains $\overline{x_i}$. Since F is read-k, the maximum degree of a vertex in this graph is $q(k-1)$. Therefore, if the graph has N vertices, it must have an independent set of size at least $N/(q(k-1)+1)$: such an independent set could be formed by repeatedly choosing a vertex to put in the set, and then eliminating from consideration all of the at most $q(k-1)$ neighbors of the vertex. On the other hand, the size of any independent set in this graph can be at most j since F is a satisfy-j DNF (any independent set of vertices corresponds to terms that can all be simultaneously satisfied). Therefore, N, the number of terms of length at most q, is at most $j(q(k-1)+1) \leq jkq$. □

The next lemma is a variant of Lemma 8, which is easier to use in our arguments to follow.

LEMMA 12. *If a is a satisfying assignment for an RkSj formula F, then either some term satisfied by a is a $4kj$-variant of $\mathit{sensitive}(a)$ or else there is a set of literals U such that $|U| \geq 2kj$ and for all $\ell \in U$, the assignment $b = \mathit{flip}(a,\ell)$ is a positive example satisfying a strict subset of the terms of F satisfied by a.*

Proof. Let T be the set of terms satisfied by a. If some term in T is a $4kj$ variant of $\mathit{sensitive}(a)$ then we are done. Otherwise, let t be any term in T, and let $W = \{w_1, w_2, \ldots, w_{4kj}, \ldots\}$ be the set of at least $4kj$ literals that are in T but are not

Algorithm *Learn-RkSj*:

1. $H \leftarrow \phi$.
2. $b \leftarrow EQ(H)$.
3. if $b =$"true" then stop and output H.
4. Else, if b is a negative counterexample then for every term $t \in H$ such that $t(b) = 1$ delete t from H.
5. Else (if b is a positive counterexample) then
   (a) $S \leftarrow$ *find-useful-example*(b). /* This procedure returns a set of examples. */
   (b) For each $a \in S$, $H \leftarrow H \cup$ *produce-terms*(a).
6. GOTO 2.

FIG. 1. *Algorithm Learn-RkSj.*

in *sensitive*(a). By Lemma 4, for at least $2kj$ literals in W there does not exist a term in F almost satisfied with respect to it. So for at least $2kj$ literals ℓ, the assignment *flip*(a, ℓ) does not satisfy any term not satisfied by a. As such an assignment does not satisfy the term t, it satisfies a strict subset of T. □

5. The learning algorithm. On a positive counterexample, the algorithm (with high probability) adds to its hypothesis a polynomial number of terms, one of which is an implicant of the target RkSj DNF formula. On a negative counterexample the algorithm deletes terms from the hypothesis that are satisfied by that example (and therefore are not implicants of the function). The number of queries will be bounded by a polynomial in n plus the number of terms in the target RkSj DNF formula. The enumerative approach discussed earlier results in a polynomial-time algorithm only when k and j are constants, because the number of neighbors and variant terms to be enumerated is $O(n^{O(kj)})$. We avoid the enumerative approach and obtain an algorithm tolerating larger (nonconstant) values of k and j, by implementing a probabilistic search in the spirit of [BR92]. We show the learnability of RkSj DNF for $k \cdot j = O(\frac{\log n}{\log \log n})$.

Our algorithm has the following high level structure (see Figure 1), which we describe here. The low level of the algorithm is described in the next two subsections.

Let F be the target function, and $F = t_1 \vee t_2 \vee \cdots \vee t_s$ its RkSj representation. (There may be more than one such representation; in such a case we fix one of them just for the purpose of the analysis.) The main tools in the algorithm are Procedures *produce-terms* and *find-useful-example* that together, given a positive counterexample b to the current hypothesis, find a set of terms such that with high probability one of them (call it t) is an implicant of F (in fact, t will be a *prime* implicant) and is implied by some term t_i satisfied by b. Note that such a term t is satisfied by all the assignments that satisfy t_i, but not by any negative assignment of F. Thus finding t is as good as (or even better than) finding the term t_i.

The disjunction of all the terms found in this way is added to the hypothesis of the algorithm, with which we now ask an equivalence query (step 2). A negative counterexample allows us to throw out from the hypothesis terms that do not imply F. A positive counterexample satisfies a term $t_i \in F$ we have not yet found, and we use it to run again Procedures *find-useful-example* and *produce-terms*.

5.1. Producing terms. We now describe the way the procedure *produce-terms* works given a positive example x of F. In our analysis, we will assume that example

Procedure *produce-terms*(x)**:**

1. Find $sensitive(x)$ (using n membership queries).
2. $L \leftarrow \emptyset$.
3. Repeat the following $m_1 = 2^{6kj} \ln(12kjs/\delta)$ times:
   Pick a random y that agrees with $sensitive(x)$ (that is, for each literal in $sensitive(x)$ we take $y_i = x_i$ and for any other literal, not in $sensitive(x)$, the corresponding bit is selected randomly). If y is negative (a membership query) then find all literals ℓ satisfied by y (and not in $sensitive(x)$) such that $flip(y, \ell)$ is a positive example. Put all those literals into L.
4. If $|L| \leq kjq^2$, then for each subset L' of L of size at most $4kj$, add to the output the term $sensitive(x) \cup L'$.

FIG. 2. *Procedure produce-terms.*

Procedure *find-useful-example*(x)**:**

Do the following $m_2 = 2^{j+1} \log(3s/\delta)$ times and produce the union of the outputs.

1. Let $x^{(0)} = x$.
2. For $i = 1, 2, \ldots, m_3$, for $m_3 = \frac{n}{2k} \ln(2j2^j)$, let $x^{(i)}$ be a random positive neighbor of $x^{(i-1)}$.
3. Output $\{x^{(0)}, x^{(1)}, \ldots, x^{(m_3)}\}$.

FIG. 3. *Procedure find-useful-example.*

x has the property that some term t_1 of F satisfied by x (1) is not in our hypothesis and (2) is a $4kj$-variant of $sensitive(x)$. We call such an example x a *useful* example. If F is a disjoint DNF, then any positive counterexample x will be useful by the special case of Lemma 8 with $j = 1$. For general RkSj DNF, we use Procedure *find-useful-example* (described in section 5.2) to find an example with these properties. Procedure *produce-terms* is described in Figure 2, where $q = c \log^2 n$ for some constant c.

Our goal is to find an implicant of F that is implied by t_1. In fact, we will find a "small" collection of potential terms such that one of them will be such an implicant. We start by finding $sensitive(x)$, and we then fix those literals for the remainder of the procedure. After fixing (projecting onto) those literals, we still have an RkSj DNF, but t_1 has at most $4kj$ literals by our assumption. Let $t = \ell_1 \ell_2 \cdots \ell_r$ $(r \leq 4kj)$ be some minimal (prime) implicant of F implied by t_1. Note that this means that for each $i \in \{1, 2, \ldots, r\}$ the term $t^{(i)} = \ell_1 \cdots \ell_{i-1} \overline{\ell_i} \ell_{i+1} \cdots \ell_r$ is *not* an implicant of F, because then we could remove ℓ_i from t and obtain an implicant, contradicting the primality of t.

The idea is that we create a set of literals L that with high probability has the following two properties: (1) L has all the literals of t, and (2) L has size poly(kj). This is done in step 3. If we have such a set then we are done: given the set L, we can look at all $\sum_{i=0}^{4kj} \binom{|L|}{i} = |L|^{O(kj)}$ small subsets (step 4), and one of them will be t.

Recall that s is the number of terms in the RkSj representation of F, and let $\delta > 0$ be a constant confidence parameter.

CLAIM 13. *If x is a useful example and t_1, t are as described above, then the set L produced in step 3 of Procedure produce-terms contains all the literals in t with probability at least $1 - \delta/3s$.*

Proof. Consider a literal ℓ_i in t. In a random y, there is a $1/2^r$ chance that literals

$\ell_1, \ldots, \ell_r$ are set so that $t^{(i)}$ is satisfied. Also,

$$\mathbf{Prob}_{y \in \mathcal{U}}[y \text{ is negative} \mid y \text{ satisfies } t^{(i)}] \geq 1/2^{2kj}.$$

The reason is that if we project variables in t to satisfy $t^{(i)}$, then we are left with an RkSj DNF which is not identically true, since $t^{(i)}$ is not an implicant of F. So, we can apply Lemma 10. For such an example y, which is negative and satisfies $t^{(i)}$, we know that $\mathit{flip}(y, \ell_i)$ is a positive example (as it satisfies t) and hence ℓ_i will be inserted into L. So, the probability that a single y "discovers" ℓ_i is at least $1/2^{r+2kj} \geq 1/2^{6kj}$. A standard calculation shows that if we repeat the loop (step 3) at least $m_1 = 2^{6kj} \ln(12kjs/\delta)$ times then we find *all* $r \leq 4kj$ literals of t, with probability at least $1 - \delta/3s$. (Note that the number of repetitions, m_1, is polynomial as long as $kj = O(\log n)$.) □

Claim 14. *Let $q = c \log^2 n$, where c is a constant that depends on the constant δ, and assume that $kj = O(\log n)$. With probability at least $1 - \delta/3s$ we get $|L| \leq kjq^2$.*

Proof. Any literal found in step 3 of *produce-terms* must be in *some* term. So, consider terms of size at most q. By Lemma 11 there are at most kjq such terms. The number of literals that those terms can contribute to L is at most kjq^2. Now consider a term of size greater than q. With high probability, it will not contribute *any* literals to L. The reason is that in order for such a term to "matter" (i.e., for some $\mathit{flip}(y, \ell)$ to satisfy that term) it must be that y satisfies all but one literal in that term. The chance that y does so is at most $n/2^q$ which, by the choice of q, is smaller than any polynomial fraction. Since the algorithm repeats in step 3 a polynomial number of times (using here that $kj = O(\log n)$), throughout the entire algorithm, with high probability, *no* example $\mathit{flip}(y, \ell)$ satisfies *any* of those terms. By choice of c this probability can be made larger than $1 - \delta/3s$. □

Notice that by Claims 13 and 14, with high probability at least one of the terms output in step 4 of Procedure *produce-terms* is the desired term t, and in addition at most $(kjq^2)^{4kj}$ terms are output in step 4 total. This number of terms is polynomial in n as long as $kj = O(\log n / \log\log n)$.

5.2. Finding a useful positive example. Procedure *produce-terms* in the previous section requires a useful example x. Namely, some term t_1 of F (1) is not in our hypothesis and (2) is a $4kj$-variant of $\mathit{sensitive}(x)$. In this section we show how to produce, given a positive counterexample x to our hypothesis, a polynomial-sized set of examples such that with high probability at least one is useful in this sense. The procedure is described in Figure 3. The idea behind Procedure *find-useful-example* is as follows. Suppose that our given positive example x satisfies some number of terms of F (none of them appears in our hypothesis). Lemma 12 states that either:

(A) all but $4kj$ of the literals in one of those terms are sensitive (i.e., x is already useful), or else,

(B) there exist at least $2kj$ literals $u_1, \ldots, u_{2kj}$ such that flipping u_i will cause the example to remain positive but satisfy a strict subset of those terms and no others.

We also know, by Lemma 4, that $|Y(x)| \leq 2kj$, and therefore there are at most $2kj$ literals that when flipped cause the example to satisfy some additional term.

What this means is that either example x is already useful, or else out of the n literals, at least $2kj$ are "good" in that flipping them makes our example satisfy a strict subset of the original set of terms, at most $2kj$ are "bad" in that flipping them makes the example satisfy some new term, and the rest are "neutral" in that either flipping them makes the example negative (which we will notice) or else flipping them

does not affect the set of terms satisfied. So, as described in Figure 3 (step 2), let $x^{(i)}$ to be a random positive neighbor of $x^{(i-1)}$ in the boolean hypercube.

CLAIM 15. *With probability* $\frac{1}{2^{j+1}}$ *at least one of the examples* $x, x^{(1)}, \ldots, x^{(m_3)}$ *is useful.*

Proof. For the moment, consider the infinite sequence $x^{(1)}, x^{(2)}, \ldots$ of positive examples. If the example x was not already useful, let's say that we are "lucky" if we flip one of the "good" literals before we flip any of the "bad" literals in this experiment (i.e., we are lucky if we flip an arbitrary number of neutral literals followed by a good literal). Notice that flipping a "neutral" literal may change the sets of good and bad literals, but our bounds on the sizes of those sets remain. So, the probability that we are lucky is at least 1/2. If this occurs, we then either have a useful positive example, or else our bounds on the sizes of the good and bad sets remain but the example satisfies at least one fewer term. Thus, with probability at least $(1/2^j)$, we continue to flip good literals before we flip any bad literals, and we reach a useful example (since by Corollary 6, if the example satisfies only one term, then it is useful).

Now for the finite sequence $x^{(1)}, \ldots, x^{(m_3)}$, we have to subtract the probability of being lucky j times, but not within the first m_3 trials. This event is included in the event "there were less than j flips of nonneutral literals within m trials." Let the latter event be E.

Consider m_3 as j blocks of $\frac{n}{2kj} \ln(2j2^j)$ trials each. The probability that we do not flip any nonneutral bit (or reach a useful example) in one block is at most $\frac{1}{2j2^j}$ (since the probability of getting a nonneutral bit is at least $\frac{2kj}{n}$ at each trial). The probability that we fail in any of the j blocks is therefore at most $\frac{1}{2^{j+1}}$. This is also a bound for the probability of the event E. Thus our total success probability is at least $\frac{1}{2^j} - \frac{1}{2^{j+1}}$. □

CLAIM 16. *With probability at least* $1-\delta/3s$, *some example in the set of examples produced by Procedure find-useful-example is useful.*

Proof. Procedure *find-useful-example* repeats the experiment described in Claim 15 for $m_2 = 2^{j+1} \ln(3s/\delta)$ times. Thus, with probability at least $1 - \delta/3s$, the experiment is successful at least once. □

5.3. Final analysis.

CLAIM 17. *With probability at least* $1-\delta$, *Algorithm Learn-RkSj uses* $O(snm_1m_2m_3)$ *membership queries and* $O(sm_2m_3(kjq^2)^{4kj})$ *equivalence queries.*

Proof. By Claim 16, each time that Procedure *find-useful-example* is invoked (on a positive counterexample), we get a useful example with probability at least $1-\delta/3s$. By Claims 13 and 14, if we have a useful example, then we find a term t that satisfies the original counterexample with probability at least $1 - 2\delta/3s$. All together, we find a good term t with probability at least $1 - \delta/s$. Therefore, if we get s positive counterexamples, with probability at least $1 - \delta$, we find prime implicants for all the terms in F, and therefore the hypothesis is logically equivalent to F.

We now count the number of queries used in this case. In each call to *produce-terms* we make n membership queries to find *sensitive*(x): one membership query for each y, and n additional queries for each y that is negative. All together Procedure *produce-terms* makes $O(nm_1)$ membership queries to create the set of literals L. It then produces (in step 4) $O((kjq^2)^{4kj})$ terms (or none, in the unlikely event that $|L| > kjq^2$). The number of calls to Procedure *produce-terms* is bounded by the number of positive counterexamples that we get in the algorithm (which is bounded by s) multiplied by the number of examples produced by Procedure *find-useful-example* on each such counterexample (this is bounded by m_2m_3). Therefore, in total we make

$O(snm_1m_2m_3)$ membership queries and produce $O(sm_2m_3(kjq^2)^{4kj})$ terms. This immediately gives a bound on the number of negative counterexamples. Hence, the number of equivalence queries is $O(sm_2m_3(kjq^2)^{4kj})$. Note that just by the read-k property of F we have $s \leq kn$. A tighter bound of $s = O(\sqrt{k^2jn})$ is given by [Mat93]. □

CLAIM 18. *Let $\delta > 0$ be a constant and $kj = O(\frac{\log n}{\log\log n})$. Algorithm Learn-RkSj runs in time* poly(n) *and finds a function logically equivalent to F with probability at least $1 - \delta$.*

Proof. By Claim 17, the algorithm succeeds with probability at least $1 - \delta$. The polynomial bound then follows, noting that m_1, m_2, and m_3 are all polynomial for $kj = O(\log n)$ and that $(kjq^2)^{O(kj)}$ is polynomial for $kj = O(\log n/\log\log n)$ (as q is polylog(n)). □

So far we considered only δ, which is a fixed constant (this is used in Claim 14 when we say that c is a constant). To generalize to arbitrary δ (e.g., δ can be a function of n), we use a standard technique as follows.

THEOREM 19. *Let F be an RkSj formula, $kj = O(\frac{\log n}{\log\log n})$, and $\delta > 0$. There is an algorithm that runs in time* poly$(n, \log\frac{1}{\delta})$ *and finds a function logically equivalent to F with probability at least $1 - \delta$.*

Proof. We run Algorithm *Learn-RkSj* $\ln\frac{1}{\delta}$ times. Every time we stop after using the number of queries stated in Claim 17, and use an equivalence query to decide if we need to run further. Clearly, the overall algorithm runs in poly$(n, \log\frac{1}{\delta})$ and succeeds with probability at least $1 - \delta$. □

As a last remark we note that it may be that the method we use here can be adapted for $kj = O(\log n)$. The only part of the algorithm that does not allow for $kj = O(\log n)$ is Procedure *produce-terms*, and in particular, step 4, which produces $O((\log n)^{O(kj)})$ terms. Finding a better sampling method or a better way to produce the terms out of the set L might solve this problem.

6. R2D DNF does not have small CNF. We present a family $\{F_n\}$ of functions that have short (poly(n)) R2D DNF formulas but require CNF formulas of size $2^{\Omega(\sqrt{n})}$. Note that this also implies that the size of any decision tree for these functions must also be superpolynomial, even without restricting the number of variable occurrences. Bshouty [Bsh95] gives an algorithm for learning functions (such as decision trees) which have both small DNFs **and** small CNFs. Our family of functions $\{F_n\}$ shows that this result does not apply here, as the size of the CNFs for even R2D DNFs may be superpolynomial. It remains open, however, whether techniques similar to those used in [Bsh95] can be applied to yield a comparable or stronger result. Such a result would rely on showing that every RkSj DNF has a small "monotone basis"; we suspect this is not the case, as even read-twice DNFs (though not disjoint) require an exponentially large monotone basis [Bsh94].

THEOREM 20. *There is a family of functions $\{F_n\}$ with polynomial-sized R2D representations but whose CNF representations require at least $2^{\sqrt{2n}} - 2(\sqrt{2n} + 1)$ clauses.*

Proof. Let $n = \binom{m+1}{2}$. We define a function F_n with $m+1$ terms each of length m on n variables $x_{i,j}$ where $1 \leq i < j \leq m+1$. The R2D representation of the function is $t_1 \vee t_2 \vee \cdots \vee t_{m+1}$, where the term t_q includes all the literals $x_{q,j}$ for $j > q$, and all the literals $\overline{x_{j,q}}$ for $j < q$. The idea is that the variable $x_{i,j}$ appears only in the ith term and the jth term, and is "responsible" for the disjointness of these two terms (since the variable appears negated in one term and unnegated in the other). For

example, for $m = 3$ the function is: $x_{12}x_{13}x_{14} \vee \overline{x_{12}}x_{23}x_{24} \vee \overline{x_{13}}\,\overline{x_{23}}x_{34} \vee \overline{x_{14}}\,\overline{x_{24}}\,\overline{x_{34}}$. The main step in the proof is the following claim.

CLAIM 21. *Every clause in a CNF representation of F_n must have $m+1$ literals.*

Given this claim, the following counting argument shows that the CNF must have a large number of clauses: the number of assignments satisfied by F_n is exactly $(m+1)2^{n-m}$, since each of the $m+1$ terms satisfies 2^{n-m} assignments (it determines the value of m out of the n variables) and the representation is disjoint. This implies that the number of assignments unsatisfied by F_n is $2^n(1-(m+1)/2^m)$. Since a clause of length $\geq m+1$ falsifies at most 2^{n-m-1} assignments, we need at least $2^{m+1}(1-(m+1)/2^m) = 2^{m+1} - 2(m+1)$ clauses to falsify all the unsatisfying assignments of F_n. As $m^2 \leq 2n \leq (m+1)^2$ the theorem follows.

Proof of the claim. Assume by way of contradiction that there is a clause of length $\alpha < m+1$ in a CNF representation for F_n. Let $C = (\ell_1 \vee \ell_2 \vee \cdots \vee \ell_\alpha)$ be that clause. We show that there exists an assignment a such that $C(a) = 0$ (and hence the value of the CNF formula on a is 0) and $F_n(a) = 1$, contradicting the assumption that C is a clause in a CNF representation of F_n.

To construct a, notice that since every *literal* appears exactly once in the R2D representation, fixing the values of $\alpha \leq m$ variables can falsify at most m terms in the DNF representation, so there is still a term t_C that can be satisfied. Therefore, we can define a to be the assignment in which all the literals in C are set to zero, all the literals in t_C are set to one, and all other literals are set arbitrarily. We thus have that $C(a) = 0$ but $t_C(a) = 1$ (and hence $F_n(a) = 1$), a contradiction. □

7. Appendix: Lemma 4 is tight. Lemma 4 plays a central role in our analysis. It gives a bound on the number of terms which are almost satisfied by an assignment a, in terms of k and j. It is natural to ask how good this bound is. The following example, due to Galia Givaty [Giv95], shows that it is essentially optimal (up to constants). More precisely, for any values of k and j we give a function $F_{k,j}$ with $k \cdot j$ variables $x_0, x_2, \ldots, x_{kj-1}$ and an assignment a such that $|Y(a)| = k \cdot j$. The function $F_{k,j}$ has kj terms:

$$t_i = \overline{x_i}x_{i+1}\ldots x_{i+k-1 \bmod kj} \qquad (0 \leq i \leq kj-1).$$

It can be easily verified that the function is read-k (the variable x_i appears only in the terms $t_{i-k+1 \bmod kj}, \ldots, t_i$). It is also satisfy-$j$, since if an assignment satisfies a term t_i it cannot satisfy any of the $k-1$ terms $t_{i-k+1 \bmod kj}, \ldots, t_{i-1 \bmod kj}$ (as they all contain the variable x_i while t_i contains $\overline{x_i}$). Now, let a be the all "1" assignment. Each term t_i is almost satisfied by a with respect to index i. Hence, $|Y(a)| = kj$.

8. Bibliographic remarks. The work presented here appeared in preliminary forms as [AP92] and [BKK+94]. The algorithm for constant k and j is from [AP92], and the extension to $k \cdot j = O(\log n / \log\log n)$, from [BKK+94]. The family of functions given in section 6 was defined in [AP92], where it was shown that the class R2D is not included in read-k decision trees. This result was later strengthened in [Aiz93] to show that the family requires decision trees of size 2^{n-2}. Its current form is from [BKK+94].

REFERENCES

[AFP92] D. ANGLUIN, M. FRAZIER, AND L. PITT, *Learning conjunctions of Horn clauses*, Machine Learning, 9 (1992), pp. 147–164.

[AHK93] D. Angluin, L. Hellerstein, and M. Karpinski, *Learning read-once formulas with queries*, J. Assoc. Comput. Mach., 40 (1993), pp. 185–210.

[AHP92] H. Aizenstein, L. Hellerstein, and L. Pitt, *Read-thrice DNF is hard to learn with membership and equivalence queries*, In Proc. 33rd Symp. on the Foundations of Comp. Sci., IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 523–532. A revised manuscript with additional results will appear in Comput. Complexity, as Aizenstein, Hegedus, Hellerstein, and Pitt, *Complexity theoretic hardness results for query learning*, to appear.

[Aiz93] H. Aizenstein, *On the Learnability of Disjunctive Normal Form Formulas and Decision Trees*, Ph.D. thesis, Dept. of Computer Science, University of Illinois, Urbana, 1993; also Technical Report UIUCDCS-R-93-1813, Dept. of Computer Science, University of Illinois, Urbana, 1993.

[AK95] D. Angluin and M. Kharitonov, *When won't membership queries help?*, J. Comput. System Sci., 50 (1995), pp. 336–355.

[Ang87] D. Angluin, *Learning k-Term DNF Formulas using Queries and Counterexamples*, Technical Report YALEU/DCS/RR-559, Department of Computer Science, Yale University, New Haven, CT, 1987.

[Ang88] D. Angluin, *Queries and concept learning*, Machine Learning, 2 (1988), pp. 319–342.

[Ang90] D. Angluin, *Negative results for equivalence queries*, Machine Learning, 5 (1990), pp. 121–150.

[AP91] H. Aizenstein and L. Pitt, *Exact learning of read-twice DNF formulas*, In Proc. 32nd IEEE Symp. on Foundations of Computer Science, San Juan, 1991, pp. 170–179.

[AP92] H. Aizenstein and L. Pitt, *Exact learning of read-k disjoint DNF and not-so-disjoint DNF*, in Proc. Annual ACM Workshop on Computational Learning Theory, Pittsburgh, PA, Morgan Kaufmann, San Francisco, 1992, pp. 71–76.

[AP95] H. Aizenstein and L. Pitt, *On the learnability of disjunctive normal form formulas*, Machine Learning, 19 (1995), pp. 183–208.

[BFJ+94] A. Blum, M. Furst, J. Jackson, M. Kearns, Y. Mansour, and S. Rudich, *Weakly learning DNF and characterizing statistical query learning using Fourier analysis*, In Proc. 26th ACM Symposium on Theory of Computing, ACM, New York, 1994.

[BFOS84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, Wadsworth International Group, CA, 1984.

[BKK+94] A. Blum, R. Khardon, A. Kushilevitz, L. Pitt, and D. Roth, *On learning read-k satisfy-j DNF*, in Proc. Annual ACM Workshop on Computational Learning Theory, ACM, New York, 1994, pp. 110–117.

[BR92] A. Blum and S. Rudich, *Fast learning of k-term DNF formulas with queries*, in Proc. 24th ACM Symposium on Theory of Computing, ACM, New York, 1992, pp. 382–389.

[BS90] A. Blum and M. Singh, *Learning functions of k terms*, In Proc. COLT '90, Morgan Kaufmann, San Francisco, 1990, pp. 144–153.

[Bsh94] N. H. Bshouty, personal communication, 1994.

[Bsh95] N. H. Bshouty, *Exact learning boolean functions via the monotone theory*, Inform. and Comput., 123 (1995), pp. 146–153; an earlier version appeared in Proc. 34th Ann. IEEE Symp. on Foundations of Computer Science, 1993.

[Giv95] G. Givaty, personal communication, 1995.

[Han91] T. Hancock, *Learning 2μ DNF formulas and $k\mu$ decision trees*, In Proc. 4th Annual Workshop on Computational Learning Theory, Santa Cruz, CA, Morgan Kaufmann, San Francisco, 1991, pp. 199–209.

[Jac94] J. Jackson, *An efficient membership-query algorithm for learning DNF with respect to the uniform distribution*, in Proc. 35th IEEE Symp. on Foundations of Computer Science, Santa Fe, NM, 1994, pp. 42–53.

[Kha94] R. Khardon, *On using the Fourier transform to learn disjoint DNF*, Inform. Process. Lett., 49 (1994), pp. 219–222.

[KLPV87] M. Kearns, M. Li, L. Pitt, and L. Valiant, *On the learnability of Boolean formulae*, In Proc. 19th Annual ACM Symposium on Theory of Computing, ACM, New York, 1987, pp. 285–294.

[KM93] E. Kushilevitz and Y. Mansour, *Learning decision trees using the Fourier spectrum*, SIAM J. Comput., 22 (1993), pp. 1331–1348; an earlier version appeared in Proc. 23rd Annual ACM Symposium on Theory of Computing, 1991.

[KR95] E. Kushilevitz and D. Roth, *On learning visual concepts and DNF formulae*, Machine Learning, 1 (1995), pp. 11–46; an earlier version appeared in Proc. ACM Workshop on Computational Learning Theory '93, ACM, New York,

[Mat93] S. Matar, *Learning with Minimal Number of Queries*, Master's thesis, University of Alberta, Canada, 1993.

[PR94] K. Pillapakkamnatt and V. Raghavan, *On the limits of proper learnability of subsets of DNF formulas*, in Proc. Annual ACM Workshop on Computational Learning Theory, ACM, New York, 1994, pp. 118–129; Machine Learning, 25 (1996), pp. 237–263.

[PR95] K. Pillapakkamnatt and V. Raghavan, *Read twice DNF formulas are properly learnable*, Inform. and Comput., 122 (1995), pp. 236–267.

[PV88] L. Pitt and L. Valiant, *Computational limitations on learning from examples*, J. Assoc. Comput. Mach., 35 (1988), pp. 965–984.

[Qui86] J. R. Quinlan, *Induction of decision trees*, Machine Learning, 1 (1986), pp. 81–106.

[Qui93] J. R. Quinlan, *C*4.5: *Programs for machine learning*, Morgan Kaufmann, San Francisco, 1993.

[Sim83] H. U. Simon, *A tight* $\Omega(\log\log n)$*-bound on the time for parallel RAM's to compute nondegenerate boolean functions*, in Lecture Notes in Computer Science 158, Springer-Verlag, New York, 1983, pp. 439–444.

[Val84] L. G. Valiant, *A theory of the learnable*, Comm. ACM, 27 (1984), pp. 1134–1142.

[Val85] L. G. Valiant, *Learning disjunctions of conjunctions*, in Proc. 9th International Joint Conference on Artificial Intelligence, Vol. 1, Los Angeles, CA, 1985, pp. 560–566.

SIAM J. COMPUT.
Vol. 27, No. 6, pp. 1531–1549, December 1998
© 1998 Society for Industrial and Applied Mathematics
002

LOG-SPACE POLYNOMIAL END-TO-END COMMUNICATION*

EYAL KUSHILEVITZ†, RAFAIL OSTROVSKY‡, AND ADI ROSÉN§

Abstract. Communication between processors is the essence of distributed computing: clearly, without communication, distributed computation is impossible. However, as networks become larger and larger, the frequency of link failures increases. The end-to-end communication problem asks how to efficiently carry out fault-free communication between two processors over a network, in spite of such *frequent* link failures. The sole minimum assumption is that the two processors that are trying to communicate are not permanently disconnected (i.e., the communication should proceed even when there does not (ever) simultaneously exist an operational path between the two processors that are trying to communicate).

We present a protocol to solve the end-to-end problem with logarithmic-space and polynomial communication at the same time. This is an exponential memory improvement to all previous polynomial communication solutions. That is, all previous polynomial communication solutions needed at least *linear* (in n, the size of the network) amount of memory per link.

Our protocol transfers packets over the network, maintains a simple-to-compute $O(\log n)$-bits potential function at each link in order to perform routing, and uses a novel technique of packet canceling which allows us to keep only *one* packet per link. The computations of both our potential function and our packet-canceling policy are totally local in nature.

Key words. end-to-end communication, space complexity, unreliable networks

AMS subject classifications. 68M10, 68Q22, 68Q25

PII. S0097539795296760

1. Introduction.

In this paper we address the problem of *communication* in distributed systems: how can two processors (a sender and a receiver) communicate over an unreliable communication network? This question was considered in many different settings, which can be divided into two groups depending on the *frequency* of link failures.

Communication during infrequent faults. If link failures are "infrequent," then after each failure a new communication path between the two communicating processors can be computed, which will be operational until the next fault. That is, in case failures occur *rarely*, it is possible, upon a failure, to "reset" the network and compute a new path between the communicating processors (e.g., [Fin79, AAG87, AS88, AAM89, AGH90]). In a related model, a communication path can also be computed in a "self-stabilizing" manner (e.g., [Dij74, AKY90, KP90, APV91, AV91, AKM+93, AO94, IL94]), which essentially means that after faults stop *for a sufficiently long period of time*, the protocol "stabilizes" to its correct behavior (i.e., establishes a new path). We emphasize that both the above reset and self-stabilizing solutions work *only if faults are not too frequent*, as they require that a message be transmitted over the computed path.

* Received by the editors December 18, 1995; accepted for publication (in revised form) August 30, 1996; published electronically June 3, 1998. An early version of this paper appeared in *Proc. of the 27th ACM Symp. on the Theory of Computing* (*STOC*), 1995, pp. 559–568.
http://www.siam.org/journals/sicomp/27-6/29676.html

† Department of Computer Science, Technion, Haifa 32000, Israel (eyalk@cs.technion.ac.il, http://www.cs.technion.ac.il/∼eyalk). Part of this research was done while the author was visiting ICSI, Berkeley, CA.

‡ Computer Science Division, University of California at Berkeley, and International Computer Science Institute, Berkeley, CA 94720 (rafail@cs.berkeley.edu).

§ Department of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel (adiro@math.tau.ac.il). Part of this research was done while the author was visiting ICSI, Berkeley, CA. Current address: Department of Computer Science, University of Toronto, ON, Canada.

Communication during frequent faults. We consider a setting where failures occur frequently. The so-called end-to-end communication problem [AG88, AMS89, AGR92, AAG+97] is to deliver, in finite time, data-items from a sender processor to a receiver processor, where data-items are given in an on-line fashion to the sender and must be output in the same order, without duplication or omission at the receiver processor. This should be done even if there does not exist, at any time, a path of simultaneously active links that connects the two communicating processors. The sole assumption is that the two communicating processors are not separated by a cut of permanently failed links.[1] Solutions to the problem are evaluated according to their *communication complexity* and *space complexity*.

1.1. Frequent faults model.

Communication complexity. One possible solution to the above problem is to give data-items "unbounded sequence numbers" and to "flood" the network with each item [Vis83, AE86]. However, this solution has the drawback that the message size increases unboundedly with the number of items sent; hence, the amount of communication needed per data-item grows unboundedly with the number of data items. Recently, the study of end-to-end protocols with *bounded* communication complexity received a lot of attention [AG88, AMS89, APV91, AG91, AGR92, AAG+97]. In this paper, we concentrate on bounded (in fact, polynomial) communication-complexity protocols.

Space complexity. Another important complexity measure is the *space complexity*—the amount of space needed at the processors, per incident link. Notice that the "unbounded sequence numbers" solution requires unbounded memory as well. The question of reducing memory requirements, while maintaining efficiency, received a lot of attention in the "self-stabilizing" setting (e.g., [MOOY92, AO94, IL94]), where it was shown that, in the case of *infrequent* memory faults, small memory and communication efficiency are simultaneously attainable. In contrast, all end-to-end protocols that were efficient in terms of their communication complexity were not efficient in their space complexity: more precisely, all end-to-end communication protocols that had polynomial communication complexity required at least *linear* (in the number of processors of the network) amount of space, at each processor, per incident link. Protocols with smaller space complexity were only presented at the cost of (at least) exponential communication complexity: Afek and Gafni [AG88] give a protocol which uses logarithmic amount of space, but has exponential communication complexity, and another protocol which uses constant amount of space but has unbounded communication complexity.

1.2. Our result.

The question of whether there exists an end-to-end communication protocol with sublinear space complexity and at the same time polynomial communication complexity remained open. In this paper, we give an affirmative answer to this question: we give a protocol that has logarithmic ($O(\log n + D)$) space complexity and polynomial ($O(n^2 mD)$) communication complexity, where n and m are the number of processors and links in the network, respectively, and D is the data-item size. This is an exponential space complexity improvement over all known polynomial communication protocols. Notice that this is achieved at a slight increase of the communication complexity compared to the best known solution [AG91]. We compare our result to the previous work in the table below.

[1] See section 2 for a formal description of the model.

Paper reference	Communication complexity (total number of bits)	Space complexity (bits per incident link)
[Vis83, AE86]	**unbounded:** ∞	**unbounded:** ∞
[AG88], Alg. 1.	**unbounded:** ∞	**constant:** $O(D)$
[AG88], Alg. 2.	**exponential:** $O(D \cdot \text{exponential}(n))$	**logarithmic:** $O(\log n + D)$
[AMS89]	**polynomial:** $O(n^9 + mD)$	**polynomial:** $O(n^5 + D)$
[AGR92]	**polynomial:** $O(n^2 mD)$	**linear** : $O(nD)$
[AG91]	**polynomial:** $O(nm \log n + mD)$	**linear:** $O(n + D)$
present work	**polynomial:** $O(n^2 mD)$	**logarithmic:** $O(\log n + D)$

Our techniques and previous work. The starting point of our investigation is the Slide protocol of [AGR92][2] and the Majority algorithm of [AAF+90, AGR92]. The Slide protocol requires storing n packets per incident link, and the decision on the recency of the received item is taken only at the receiver processor (using the technique of [AAF+90]). If we wish to reduce the space per link, we can no longer afford storing n different packets, but must keep far fewer packets per link, and we can no longer afford the Majority algorithm of [AAF+90, AGR92], which collects a large number of packets arriving at the receiver and then calculates the value representing the majority among the received values. Thus, we must somehow "drop" all but several packets, and decide which to keep at each of the processors. Moreover, we must design an "online" analogue of the majority calculation, where we can "discard" packets as soon as they arrive at the receiver processor, yet manage to compute the majority value. However, if we begin to "drop" packets everywhere, is it no longer the case that the technique of deciding which packet is the correct one to output (at the receiver) still works. To overcome both difficulties, we combine two ingredients:

- a potential function that controls the flow of data-items in the network, which is an analogue to the potential function of the Slide protocol; and
- a novel data-item canceling policy which makes sure that in any processor there will be at most two distinct values of data-items per link. The same policy is used in both the intermediate processors and the receiver processor.

We show that a combination of these two techniques yields the desired result. Hence, even in the presence of frequent link failures, it is possible to achieve *both* polynomial communication and logarithmic space. As in [AMS89, AGR92], our solution has the additional benefit that it is totally local in nature. For example, the locality of Slide was used for establishing its self-stabilizing extension in [APV91]; similar local techniques were also used for various multicommodity flow problems in dynamic graphs [AL93, AL94].

1.3. Organization. Section 2 contains all the necessary background including the formal definitions of the model and the problem. Section 3 contains the description of the protocol; we start with an informal description (section 3.1) and then give a formal description (sections 3.2 and 3.3). Then, we show some of the properties of the protocol (section 3.4, with some proofs deferred to Appendix A). We conclude with its proof of correctness and complexity (section 4).

[2] A journal version of this paper, jointly with [AMS89], appears in [AAG+97].

2. Model and problem statement.

2.1. The network model. A *communication network* is associated with an undirected graph $G(V, E)$, $|V| = n$, $|E| = m$, where nodes correspond to processors and edges correspond to links of communication. We denote by $E(v)$, for $v \in V$, the set of edges which are adjacent to v. Processors are modeled as identical (except the sender and the receiver) deterministic finite state transducers with $O(\log n)$ memory per edge. (We do not require processors to have unique identifiers.) We model each *undirected* communication link as consisting of two *directed* links, delivering messages in opposite directions. Each transmission of a message is associated with a *send* event and a *receive* event; each event has its time of occurrence according to a global time, unknown to the processors. Without loss of generality, we assume that no two events occur at exactly the same time. A message is said to be *in transit* any time after its send event and before its receive event. As discussed in the introduction we would like to deal with networks whose links may fail and recover frequently. Each failure and recovery is eventually reported to both end-points of the link. When a link fails, messages that are in transit over it are lost. In the following we give a formal definition of a somewhat simpler-to-discuss model that we use in the sequel. Intuitively, the main difference between the models is that in the model we define below links never recover if they fail, and that such a failure is *not* reported to the processors. It is known that this simpler model, which was also used in [AG88, AMS89, AGR92], does not cause any reduction in power (for completeness, we prove this fact in section 2.4). In our model each *directed* link satisfies the following properties.

- Each link has *constant* capacity; that is, the protocol must obey the rule that only a constant number of messages are in transit on a given link at any given time.
- Communication over links obeys the *FIFO* (first in, first out) rule; that is, at any time, the sequence of messages *received* over the link is a prefix of the sequence of messages sent over the link.
- Communication is *asynchronous*; that is, there is no a priori bound on message transmission delays over the links.

A directed link is called *nonviable* if starting from some message and on it does not deliver any message; the transmission delay of this message and any subsequent message sent on this link is considered to be infinite. The sequence of messages received over the link is in this case a *proper* prefix of the sequence of messages sent. Otherwise, the link is *viable*. An undirected link is *viable* if both directed links that it consists of are viable.

We say that the sender is *eventually connected* to the receiver if there exists a (simple) path from the sender to the receiver consisting entirely of viable (undirected) links. Note that if there is a cut of the network, disconnecting the sender from the receiver, such that all the directed links crossing the cut are nonviable links, then eventually it becomes impossible to deliver messages from the sender to the receiver.

Remark. Notice that we model an undirected graph as a bi-connected directed graph. Hence we assume that either both directed links are viable or both are not viable. In this case, the above assumption about the eventual connectivity of the sender and the receiver is in fact the minimal possible for communication between them. On the other hand, in the model of *directed* graphs, it could be the case that there is a *directed* path from the sender to the receiver, and maybe (another) directed path from receiver to sender, yet all undirected edges are nonviable. We do not consider such (more difficult) case, and are in fact dealing only with undirected

graphs.

2.2. The end-to-end communication problem. The purpose of an end-to-end communication protocol is to establish a (directed) "virtual link" to be used for the delivery of data-items from one special processor, called the *sender*, to another special processor, called the *receiver*. Each data-item is a character of some alphabet of size 2^D; that is, each data-item can be encoded by D bits. It is required that if the sender is eventually connected to the receiver then the "virtual link" established by the protocol will be viable. This virtual link should have the same properties as a "regular" network link; namely, it should satisfy:

Safety: The sequence of data-items output by the receiver, at any time, is a *prefix* of the sequence of data-items input by the sender.

Liveness: If the sender is eventually connected to the receiver, then each data-item input by the sender is eventually output by the receiver.

A protocol for the end-to-end communication problem is given, *in an on-line fashion*, a sequence of data-items at the sender (i.e., every data-item must be delivered without the knowledge of the next data-item to be transmitted) and generates a sequence of data-items at the receiver that obey the safety and liveness properties.

2.3. The complexity measures. We consider the following complexity measures.

Communication: The number of bits transferred in the network in the worst case, per data-item delivered. More precisely, the total number of bits sent in the worst case in the period of time between two successive data-item output events at the receiver (measured in terms of n, m, and D).

Space: The maximum amount of space per incident link required by a processor's program throughout the protocol (measured in terms of n, m, and D).

In addition, we require that the local computation of the processors be polynomial for each send/receive event at each processor (in fact, our protocol uses a constant number of computational steps per event).

2.4. Relations to other models. The model described above is called the "∞-delay model" in [AG88] and the "fail-stop model" in [AM88]. As mentioned, we would like to deal with networks where links fail and recover frequently; in such *dynamic* networks, links may fail and recover many times (yet processors never fail) (see [AAG87]), and each failure or recovery of a network link is eventually reported at both its end-points by some underlying link protocol. This model should be contrasted with the self-stabilizing model (e.g., [Dij74]), where both processors and links can start in an inconsistent state, but it is assumed that they never fail after the computation begins. As was discussed in the introduction, the self-stabilizing model corresponds to *infrequent memory faults* and is incomparable to the model of *frequent link failures* addressed here. The question of how, in addition to frequent failures of links, one can allow inconsistent initial memory states was addressed in the end-to-end setting in [APV91, DW93].

As pointed out in [AG88], one can design protocols in the *fail-stop* model and convert them to the dynamic model. For this, a message to be forwarded on a link is stored in a buffer until the link recovers and the previously sent message has been delivered. A protocol similar to the data-link initialization protocol of [BS88] is used to guarantee that no message is lost or duplicated. Each link in the dynamic network that fails and never recovers for a long enough period to allow the delivery of a message is represented by a nonviable link. Note that the only space used by the

above transformation is for storing the buffers (i.e., per each link, it is the capacity of the link times the size of the longest message).[3]

3. The protocol.

3.1. High-level description. Our starting point is a linear space, yet simple, solution of [AGR92]. Their solution combines (as black-boxes) two components: the Slide protocol and the Majority algorithm. Before explaining our protocol, we give a short overview of the approach taken in [AGR92]. The Slide is used to transfer tokens (packets containing a data-item) in the network. This is done by letting each processor maintain a stack of tokens for each of its links. On each link, if active, tokens move from a larger stack of tokens to a smaller stack of tokens. The sender always has a large stack of tokens, so it only sends tokens out, and the receiver always has an empty stack, hence it only receives tokens. The Majority algorithm enables the receiver to decide, by collecting a sufficiently large number of tokens (containing data-items) and taking the majority value, what is the sequence of data-items sent by the sender. It is proved in [AGR92] that the combination of these two components yields a polynomial communication solution to the end-to-end problem.

In order to make the space requirements of the protocol logarithmic per edge, we can no longer use the Slide as a method to establish the virtual link between the sender and the receiver and we can no longer use the Majority algorithm. The reason is that the Slide needs a lot of space to store the stack of tokens (for each link), and the Majority algorithm needs even more space to collect the tokens in order to decide on the correct value to output. To overcome this, we introduce the following procedure.

- While transferring packets from the sender to the receiver, our protocol "cancels" some of these packets *both* en route and upon their arrival at the receiver processor. More precisely, it replaces some packets by "nil" packets.
- The canceling policy is designed so as to guarantee that at most two different packet-values are to be stored at each processor (including the receiver) at any given time—a "real" packet and a "nil" packet. We use a potential function that only *counts* the number of packets of both types, rather than storing all of them, and use this function to control the flow of tokens. We prove that two counters (of $\lceil \log n \rceil$ bits each) per link are sufficient for this. Altogether, we use only two counters per link and store only one data-item per processor.

Note that, usually, protocols that change the values of packets are undesirable. However, our protocol changes these values in a very restricted way—it may only replace a "real" packet by a "nil" packet. Our "canceling" policy does not effect the routing properties of the Slide protocol: it guarantees that if the sender and the receiver are eventually connected, then tokens will be transferred from the sender to the receiver. Moreover, our protocol guarantees an upper bound on the number of tokens that are in the network at any given time. Denote this upper bound by $\mathcal{C}$. For each data-item to be sent, the sender transmits to the receiver $2\mathcal{C}+1$ packets (tokens) that contain that data-item. The receiver *in an on-line, space-efficient fashion*, "collects" the same number (i.e., $2\mathcal{C}+1$) of tokens (some of which, but no more than $\mathcal{C}$, may be old tokens remaining in the network from previous transmissions) and outputs the data-item that represents the majority amongst the tokens received, *ignoring the nil tokens*.[4] We emphasize that the receiver computes this majority *without storing the*

[3] The communication is increased by a multiplicative factor of the number of failures of the link.
[4] For the first data-item only $\mathcal{C}+1$ tokens are collected.

tokens; rather, it does so by using the same "canceling policy" as in the intermediate processors.

As is clear from the above description, the heart of our protocol is the new (local) "canceling policy," described below. Whenever a token arrives into a processor on a given edge e we do the following.

- If it is a nil-token, then the counter of nil-tokens of edge e is augmented by one.
- If it is a "real" data-token and the data-item is identical to the data-item currently stored in the processor, then the data-tokens counter of edge e is augmented by one.
- If it is a "real" data-token and the data-item is different from the data-item currently stored in the processor, then the arriving token, and one data-token already accounted for in the processor are both "canceled" and become two nil-tokens (that is, the counter of nil-tokens of edge e is augmented by one, and for some edge e', whose data-tokens counter is greater than 0, the counter of nil-tokens is augmented by one, and the counter of data-tokens is decreased by one). If as a result there are no more "real" data-tokens accounted for in the processor (over all edges), we erase the current data-item stored in the processor.
- If it is a "real" data-token and there is no data-item currently stored in the processor, we store the arriving data-item as the current one, and set the counter of data-tokens of e to 1.

The essential idea of the above "canceling" policy is that from the point of view of the majority calculation done by the receiver, the above cancelation of two data-items into nil-tokens has only a minor effect; the receiver only needs to ignore the nil-tokens. Intuitively, since one of the properties of the old Majority algorithm is that, without these cancelations, the "correct" data-item would have more than half the tokens in any block of $2\mathcal{C}+1$ tokens then, at worst, if any of the canceled data-items is the correct one, then the other canceled data-item is an old one. Therefore, the majority of the correct data-items is maintained, while storing only one data-item per processor, and two counters per link.

3.2. A formal description of the protocol. We begin by describing the data-structures and messages used by our protocol. The protocol uses the following types of messages:

TOKEN **messages:** to carry data-items (either "real" data-items or the "nil" data-item).

TOKEN_LEFT **messages:** to announce over a link e that a token, accounted for in the counters of link e, has been sent away.

ACK_TOKEN **and** ACK_TOKEN_LEFT **messages:** are used to acknowledge the arrival of a TOKEN message or a TOKEN_LEFT message, respectively.

The following data is stored at each processor v:

- A variable `current_message` that stores a single data-item to be duplicated and sent in TOKEN messages.
- For each incident link e, there are two counters, `message_tokens[e]` and `nil_tokens[e]`. The variable `message_tokens[e]` records a value associated with the number of tokens that arrived on e carrying the value `current_message`; however, the value stored is not exactly this number, as sometimes a stored token is converted from a message-token into a nil-token, at which event the value of `message_tokens[e]` is decreased by one and the value of `nil_tokens[e]` is increased by one.
- For each incident link e, a variable `bound[e]` that stores an estimate on the sum of the above counters on the other side of the link. This bound is initialized to 1, incremented by one every time a token is sent over the

outgoing link, and decremented by one every time a TOKEN_LEFT message is received over the corresponding incoming link.

- For each incident link $e = (v, u)$, a counter called `tokens_left_pending[e]`, which counts the number of TOKEN messages that were sent from v (possibly on other links) on "the account" of e (i.e., caused the counters of e to decrease), but have not yet been reported as such to u. This counter is initialized as 0, incremented by one when a token leaves the counters of e, and decremented by one when a TOKEN_LEFT message is sent over e.
- For each incident link e, a flag `free_for_token` indicating whether an ACK_TOKEN message has already been received for the previous TOKEN message sent on e, and a flag `free_for_token_left`, indicating whether an ACK_TOKEN_LEFT message has already been received for the previous TOKEN_LEFT message sent on e.

The sender and the receiver store some additional data. The sender stores the last message it has received to transmit (`current_input`), together with a count of how many such packets are still to be sent (`left_to_send`). In addition, it stores part of the above stated data relative to an additional "virtual link" through which it introduces new tokens into the network. Although we call it a virtual *link*, in what follows it is not included in the set E. The receiver maintains a counter storing the number of packets it received since the time of the last output event (`count`), and another counter storing the number of packets that contained the message stored in `current_message` (`current_message_count`). It also maintains a single-bit flag `first_item`.

Throughout the proofs we assume a global time, unknown to the processors, and we denote the value of variables in a processor at a given time by a subscript of the processor and a superscript of the time (e.g., $\mathcal{X}_v^t$). We also use the following notation to count the number of different messages on a given link at a given time: Let $tokens^t_{u\to v}$ be the number of TOKEN messages in transit from u to v at time t. Let $tokens\_left^t_{u\to v}$ be the number of TOKEN_LEFT messages in transit from u to v at time t.

In our protocol the nodes do not store tokens, but merely store a single message-value and several counters. However, for our proofs we sometime use the terminology that tokens are stored, or present, in the nodes. The analogy is straightforward: when a counter counts k nil-tokens, we sometimes refer to it as if the node stores k nil-tokens, etc. This enables us to talk about the "number of tokens in the network," rather than about the "sum of values of the counters in the network," and make our arguments more readable.

3.3. The code. In this section, we present the code of our protocol. Following [AMS89, AGR92], the presentation of the code is based on the language of *guarded commands* of Dijkstra (see [DF88]), where the code of each process is of the form

$$\textbf{Select}\ \ G_1 \to A_1 \,\square\, G_2 \to A_2 \,\square\, \ldots \,\square\, G_l \to A_l\ \ \textbf{End Select}.$$

The code is executed by repeatedly selecting an arbitrary i from all guards G_i which are true, and executing A_i. Each guard G_i is a conjunction of predicates.

The predicate **Receive** M is true when a message M is available to be received. If the statements associated with this predicate are executed, then prior to this execution the message M is actually received. The message may contain some values that are assigned, upon its receipt, to variables stated in the **Receive** predicate (e.g., **Receive** TOKEN(data)). The command **Send** M on e sends the message M on the link e.

We present the code of a regular processor (Figure 1), which is every processor except the sender and the receiver. The code of the receiver is presented separately (Figure 3). The sender has an additional "virtual link" (which does not belong to the set E) through which new tokens are introduced into the network. This virtual link is denoted in the code by I. The code of the sender is composed of two parts: the code of a regular processor, and additional special code (Figure 2). Upon initialization, the sender executes the two initialization procedures (of a regular processor and the additional one), and then regards the two select commands as a unified one.

3.4. Properties of the protocol. In this section we present properties of the protocol which later allow us to prove its correctness and complexity. The structure of the proof follows the structure of the proofs in [AGR92], and some of the proofs are analogous to those of [AMS89, AGR92]. We first state the following technical lemmas (we postpone their proofs to Appendix A).

Consider an edge $e = (u, v) \in E$. The following lemma relates the estimate $bound[e]_u^t$ which u has on the number of tokens that are stored in v (in the counters corresponding to the same edge e). It shows that this estimate essentially equals the actual number of tokens stored (i.e., $message\_tokens[e]_v^t + nil\_tokens[e]_v^t$) plus those that have left, but were not yet reported as such, and those which are still in transit (i.e., $tokens_{u\to v}^t + tokens\_left_{v\to u}^t$).

LEMMA 1. *At any time t, and for any $e = (u, v) \in E$,*

$$\begin{aligned} bound[e]_u^t - 1 = {} & message\_tokens[e]_v^t + nil\_tokens[e]_v^t \\ & + tokens\_left\_pending[e]_v^t + tokens_{u\to v}^t + tokens\_left_{v\to u}^t \ . \end{aligned}$$

The next lemma gives the main intuition for the progress in the protocol.

LEMMA 2. *Consider a* TOKEN *message sent from processor u to processor v, on link $e = (u, v)$. Let e' be the link whose counter at processor u decreased when the message was sent. If just before the message is sent $message\_tokens[e']_u + nil\_tokens[e']_u = i$ and just after it is received $message\_tokens[e]_v + nil\_tokens[e]_v = j$, then $j < i$.*

Since all the tokens in the network are either "stored" in the processors or in transit over links, the following lemma and corollary will allow us to give a bound on the capacity of the network (property ($\mathcal{P}1$) below).

LEMMA 3. *At any time t and for any $e = (u, v)$,*

$$bound[e]_u^t \le n \ .$$

COROLLARY 4. *The following hold for any time t.*

(1) *For any $e = (u, v) \in E$,*

$$message\_tokens[e]_v^t + nil\_tokens[e]_v^t + tokens_{u\to v}^t \le n \ ,$$

$$tokens\_left\_pending[e]_v^t \le n \ .$$

(2) *For $e = I$,*

$$message\_tokens[e]_v^t + nil\_tokens[e]_v^t \le n \ ,$$

$$tokens\_left\_pending[e]_v^t \le n \ .$$

```
Select
Initialization ⟶
    C := n · (2m + 1) ;
    current_message=empty;
    for every incident link e ∈ E(v)
        bound[e]:=1; message_tokens[e]:=0; nil_tokens[e]:=0; tokens_left_pending[e]:=0;
        free_for_token[e]:=true;  free_for_token_left[e]:=true;
□
Receive TOKEN_LEFT on e ⟶
    bound[e]:=bound[e]-1;
    Send ACK_TOKEN_LEFT on e;
□
Receive TOKEN(data) on e ⟶
    if (data=nil) then
        nil_tokens[e]:=nil_tokens[e]+1;
    else
        if (current_message=empty or current_message=data) then
            message_tokens[e]:=message_tokens[e]+1;
            current_message:=data;
        else
            nil_tokens[e]:=nil_tokens[e]+1;
            for some e′ ∈ E(V) s.t. message_tokens[e′] >0
                nil_tokens[e′]:=nil_tokens[e′]+1;
                message_tokens[e′]:=message_tokens[e′] -1;
            if (for every e message_tokens[e]=0) then current_message:=empty;
        endif
    endif
    Send ACK_TOKEN on e;
□
Receive ACK_TOKEN on e ⟶
    free_for_token[e]:=true;
□
Receive ACK_TOKEN_LEFT on e ⟶
    free_for_token_left[e]:=true;
□
∃e, e′ ∈ E(v) s.t. nil_tokens[e′]+message_tokens[e′] > bound[e] and
                   free_for_token[e]=true ⟶      /* possibly e′ = e */
    if (nil_tokens[e′] > 0) then
        Send TOKEN(nil) on e
        nil_tokens[e′]:=nil_tokens[e′] - 1;
    else
        Send TOKEN (current_message) on e;
        message_tokens[e′]:=message_tokens[e′] - 1;
    endif
    free_for_token[e]:=false;
    bound[e]:=bound[e]+1;
    tokens_left_pending[e′]:= tokens_left_pending[e′] + 1;
□
∃e ∈ E(v) s.t. tokens_left_pending[e] > 0 and free_for_token_left[e]=true ⟶
    send TOKEN_LEFT on e;
    free_for_token_left[e]:=false;
    tokens_left_pending[e]:=tokens_left_pending[e]- 1;
End Select
```

FIG. 1. *Code of ordinary processor v.*

Select
Initialization $\longrightarrow$
    $\mathcal{C} := n \cdot (2m+1)$;
    current_input=`nil`;
    left_to_send:=0;
    message_tokens[I]:=0;
    nil_tokens[I]:=0;
    tokens_left_pending[I]:=0;
$\square$
left_to_send = 0 $\longrightarrow$
    current_input := `input data-item`;
    left_to_send:= $2\mathcal{C}+1$;
$\square$
message_tokens[I] < n **and** left_to_send > 0 $\longrightarrow$
    message_tokens[I]:=message_tokens[I] +1;
    left_to_send:=left_to_send - 1;
$\square$
$\exists e \in E(v)$ s.t. message_tokens[I] > bound[e] **and** free_for_token[e]=`true` $\longrightarrow$
    Send TOKEN (current_input) on e;
    message_tokens[I]:=message_tokens[I] - 1;
    free_for_token[e]:=`false`;
    bound[e]:=bound[e]+1;
End Select

FIG. 2. *Additional code for the sender.*

We use the above lemmas to prove the following theorem.

THEOREM 5. *The protocol has the following properties.*

($\mathcal{P}1$) *At any time t, the number of tokens in the network is bounded by $\mathcal{C} = n \cdot (2m+1)$.*

($\mathcal{P}2$) *Consider a time interval in which k_{new} new tokens are inserted into the network. During this time interval, at most $O(n^2 m + k_{\text{new}} \cdot n)$ TOKEN messages are sent in the network.*

($\mathcal{P}3$) *If the sender and the receiver are eventually connected, the sender will eventually insert a new token into the network.*

Proof. Every token in the network is either in transit on a link or accounted for in some counter at a processor. For every edge $e = (u, v) \in E$ we consider its two directions. For the direction from u to v (v to u) Corollary 4 states that at any given time the number of tokens in transit from u to v (v to u) plus the number of tokens accounted for at the counters of e at node v (node u) is at most n. Thus we get at most $n \cdot 2m$ tokens. At most n additional tokens can be accounted for at the counters of the "virtual link" I, altogether $n \cdot (2m+1)$ tokens. This concludes the proof of property ($\mathcal{P}1$).

We can now also prove property ($\mathcal{P}2$). Define the following potential function. For any processor v and for any incident link $e = (v, u)$, and for $e = I$ if $v = s$, denote $J^t(v,e) = message\_tokens[e]_v^t + nil\_tokens[e]_v^t$ and let $H^t(v,e) = \sum_{k=1}^{J^t(v,e)} k = \binom{J^t(v,e)}{2}$. Let P^t be the set of TOKEN messages in transit at time t. For $p \in P^t$ let t' be the time just before the TOKEN message p was sent, say from v to u. If e' is the link whose counters accounted for the token sent, then we define $T^t(p) = message\_tokens[e']_v^{t'} + nil\_tokens[e']_v^{t'}$ (that is, the token "carries" the "number" of tokens in v, at link e', just before it left this processor). Denote the sender by S. The potential function is

```
Procedure check_and_output
    if first_item and count=C + 1 then /* first data-item */
        output(current_message);
        current_message := empty;
        count:= 0;
        current_message_count:= 0;
        first_item:=false;
    else
        if (not first_item) and count= 2·C + 1 then /* all other data-items */
            output(current_message);
            current_message := empty;
            count := 0;
            current_message_count:= 0;
        endif
    endif
End Procedure
Initialize ⟶
    C := n · (2m + 1) ;
    current_message := empty;
    current_message_count:=0;
    count:=0;
    first_item:=true;
    for every incident link e ∈ E(u)
        bound[e]:=0 ;
        message_tokens[e]:=0;
        nil_tokens[e]:=0;
        tokens_left_pending[e]:=0;
        free_for_token_left[e]:=true;
□
Receive TOKEN(data) on e ⟶
    count:=count + 1;
    if (data ≠ nil) then
        if (current_message=empty or current_message=data) then
            current_message_count:=current_message_count+1;
            current_message:=data;
        else
            current_message_count:=current_message_count -1;
            if (current_message_count=0) then current_message:=empty;
        endif
    endif
    tokens_left_pending[e]:= tokens_left_pending[e] + 1;
    Send ACK_TOKEN on e;
    call check_and_output;
□
Receive ACK_TOKEN_LEFT on e ⟶
    free_for_token_left[e]:=true;
□
∃e ∈ E(u) s.t. tokens_left_pending[e] > 0 and free_for_token_left[e]=true ⟶
    send TOKEN_LEFT on e;
    free_for_token_left[e]:=false;
    tokens_left_pending[e]:=tokens_left_pending[e]- 1;
End Select
```

FIG. 3. *Code of the receiver u .*

$$\Phi^t = H^t(S,I) + \sum_{e=(u,v)\in E} [H^t(v,e) + H^t(u,e)] + \sum_{p\in P^t} T^t(p).$$

This potential function may change upon one of the following three events.

1. A new token enters the network—the potential function increases by at most n.
2. A token is sent—the potential function does not change, since the relevant H function decreases by exactly the value of the relevant new T function that is added to the sum.
3. A token is received—the potential function decreases by at least 1. This follows from Lemma 2, as the value of the function T that is extracted from the sum, is larger by at least 1 than the sum $message\_tokens + nil\_tokens$ at the accepting link, which is the exact increase in the corresponding function H.

Since by Corollary 4,

$$message\_tokens[e]_v^t + nil\_tokens[e]_v^t \le n,$$

and since there is at most one TOKEN in transit on each link at any given time, the value of Φ is at most $2m \cdot (\binom{n}{2} + n) + \binom{n}{2}$; also, Φ is clearly always nonnegative. For each token received, Φ decreases by 1, and it can increase only upon the entry of a new token to the network and by at most n. Thus if in a given time interval k_{new} new tokens are introduced, then an upper bound on the number of token receipts in this time interval is $2m \cdot (\binom{n}{2} + n) + \binom{n}{2} + k_{\text{new}} \cdot n$, since otherwise the potential function Φ would become negative. Since by the code a TOKEN message is sent on link e only after an acknowledgment on the previous TOKEN message is received, the number of TOKEN messages sent in the time interval is at most $2m$ more than the number of TOKEN messages received in the time interval. Thus, we get property ($\mathcal{P}2$).

We now turn to the proof of property ($\mathcal{P}3$). By way of contradiction, assume that t is the last time at which a new token enters the network. As a result of property ($\mathcal{P}2$) and as there is at most one TOKEN_LEFT message per TOKEN message, and one ACK message (of the corresponding type) per TOKEN and TOKEN_LEFT message, there is a time $t' \ge t$ after which no TOKEN, TOKEN_LEFT, ACK_TOKEN, or ACK_TOKEN_LEFT messages are sent. As the sender, S, and the receiver, R, are eventually connected, there is a path $R = v_0, v_1, \ldots, v_{k-1}, v_k = S$, $k < n$, such that for each $0 \le i \le k-1$, the edge $e = (v_i, v_{i+1})$ is viable; hence there is a time $t'' \ge t'$ by which all messages between v_i and v_{i+1}, in both directions, are delivered. Note that we enumerate the nodes on the path in the direction from the receiver to the sender.

Next, note that this implies that $tokens\_left\_pending[e]_{v_i} = 0$ for any i and for any time after t''. We now prove by induction on the length of the viable path from v_i to R that for any link e incident to v_i, after time t'', $message\_tokens[e]_{v_i} + nil\_tokens[e]_{v_i} \le i$. The receiver, v_0, has no tokens stored at all, thus the induction basis holds. Let $e = (v_{i-1}, v_i)$ (for $i \ge 1$), and apply the induction hypothesis to the link e in node v_{i-1}, i.e., $message\_tokens[e]_{v_{i-1}} + nil\_tokens[e]_{v_{i-1}} \le i-1$. Applying Lemma 1, and since after t'' there is no message in transit between v_{i-1} and v_i, and $tokens\_left\_pending[e]_{v_{i-1}} = 0$, we get $bound[e]_{v_i} \le i$. As $t'' \ge t'$, no token is sent after t'', but according to the code this can happen only if for any time after t'', and any e incident to v_i, $message\_tokens[e]_{v_i} + nil\_tokens[e]_{v_i} \le i$, proving the induction step.

Thus, for the "virtual link" I at the sender S, $message\_tokens[I]_S + nil\_tokens[I]_S \leq k < n$, and by the code of the sender a new token is introduced into the network, contradicting the assumption. Property ($\mathcal{P}3$) follows. □

4. Correctness proof of the protocol. In this section we prove the safety and liveness properties of the protocol.

THEOREM 6 (safety). *At any time the output of the receiver is a prefix of the input of the sender.*

Proof. Denote by $IN = (I_1, I_2, \ldots)$ the input to the sender, and by $OUT = (O_1, O_2, \ldots)$ the output of the receiver. Denote by $t_i, i > 0$, the time at which O_i is output.

By the code, the receiver outputs at t_i the value in *current_message* as the next output. We will therefore show that at t_i the value of *current_message* is I_i. For this, we consider the tokens received by the receiver in the time interval between the time data-item $i-1$ and data-item i are output. We use the following definitions.

DEFINITION 1. *Let $in^{(t,t']}$ be the number of tokens introduced into the network by the sender in interval of time $(t, t']$. Let $out^{(t,t']}$ be the number of tokens received by the receiver in the interval of time $(t, t']$.*

First note that the total number of tokens (either nil-tokens or message-tokens) received by the receiver by time t_i is

$$out^{(t_0,t_i]} = \mathcal{C} + 1 + (i-1)(2\cdot\mathcal{C} + 1).$$

By Theorem 5, the capacity of the network is $\mathcal{C}$, thus the total number of tokens sent by the sender by any time t is at most $\mathcal{C}$ more than the total number of tokens received by the receiver by the same time, t. Thus,

$$in^{(t_0,t_i]} \leq i(2\cdot\mathcal{C} + 1).$$

Since the sender sends successively $2\mathcal{C}+1$ tokens for each data item, this implies that all tokens sent by time t_i carry the value I_j for some $j \leq i$.

As to the first data-item, this guarantees that all TOKEN messages received by time t_1 carry the first data-item, and since at the beginning of the run *current_message* is initialized to `empty`, this guarantees that at time t_1 *current_message* is I_1.

To prove the claim for $i > 1$ we first show that more than half of the tokens received by the receiver in $(t_{i-1}, t_i]$ that carry a data-item when received (as opposed to nil-tokens), carry the value I_i. Since $in^{(t_0,t_i]} \leq i(2\cdot\mathcal{C}+1)$ no token carrying I_k, for $k > i$ is present in the network by time t_i. Therefore, and since $out^{(t_0,t_l]} = (l-1)(2\mathcal{C}+1) + (\mathcal{C}+1)$, any token received in $(t_{i-1}, t_i]$ is a token that is either:

- sent with value I_i (note that no such token can be received by t_{i-1}), or
- sent with value I_k, $k < i$, and not received by time t_{i-1}. There can be at most $\mathcal{C}$ such tokens, as the total number of tokens ever sent with I_k $(k < i)$, is $(i-1)(2\mathcal{C}+1)$.

The set of tokens received in $(t_{i-1}, t_i]$ is thus some subset of size $2\mathcal{C}+1$ of the above $2\mathcal{C}+1+\mathcal{C}$ tokens. We argue that in any subset of size $2\mathcal{C}+1$ of these tokens, more than half of the "real" tokens carry the value I_i. This is true along time, even after the occurrence of "cancelation" events that cause two tokens to become nil. At t_{i-1} the claim is true since no one of the $2\mathcal{C}+1$ tokens to be sent with I_i is sent yet, thus they still "carry" I_i, while there are at most $\mathcal{C}$ other tokens. Consider any cancelation event; then if the number of tokens carrying I_i is reduced as a result of one such token becoming nil, then a token carrying a value I_k, $k < i$ also becomes nil.

We now show that indeed at time t_i the value of *current_message* is I_i. Consider the sequence of $2\mathcal{C}+1$ tokens received during the interval $(t_{i-1}, t_i]$. For the purpose of

the proof we give to each token that arrives at the receiver in the time interval $(t_{i-1}, t_i]$, a number which is its number in the sequence of the tokens that arrive after time t_{i-1}. Let $1 \leq T_1 < T_2 < T_j < \cdots < T_k \leq 2\mathcal{C}+1$ be the token numbers upon whose arrival the value of *current_message* changes from `empty` to another value. Let Val_j be the new value assigned to *current_message*. We show that at time t_i *current_message* is not `empty`, and that the last "real" value assigned to it, Val_k, is I_i. Consider the set of tokens with numbers $T_j \leq l < T_{j+1}$, for $j < k$. If at T_{j+1} a new, not `empty`, value is assigned to *current_message*, then *current_message_count* was 0 at this time, which means that it has become such at the receipt of token number $T_{j+1} - p$, for some $p \geq 1$, and any token with number s, such that $T_{j+1} - p + 1 \leq s < T_{j+1}$ (if any) was a nil-token. Thus, we conclude that among the tokens $T_j \leq l < T_{j+1}$ that carry a data-item (i.e., are not nil-tokens), exactly half carry Val_j. Whatever the value Val_j is, *at most* half of the above tokens carry I_i. If upon the receipt of token $2\mathcal{C}+1$, *current_message* is `empty` (and is thus `empty` at time t_i), we have that over the whole sequence the number of tokens carrying I_i is at most half the number of tokens carrying any value. Thus, this cannot happen (i.e., *current_message* is not `empty` at time t_i). Next, assume that $Val_k \neq I_i$ (and this is the value at time t_i). Then *more* than half the tokens $T_k \leq l \leq 2\mathcal{C}+1$, that are not nil-tokens, carry value Val_k. Then, over the whole sequence the tokens carrying I_i are less than half the number of tokens carrying any value, a contradiction again. □

THEOREM 7 (liveness). *If the sender and the receiver are eventually connected, then the receiver eventually outputs any data-item input to the sender.*

Proof. If the sender inputs the ith data-item, then it tries to send $i(2\cdot\mathcal{C}+1)$ tokens (counted over the whole run). As the sender and the receiver are eventually connected, by property ($\mathcal{P}3$) all these tokens are eventually input into the network. Since the network can delay at most $\mathcal{C}$ tokens, the receiver will eventually receive $i(2\cdot\mathcal{C}+1) - \mathcal{C}$ tokens, and thus outputs the ith data-item. □

4.1. The complexity of the protocol.

LEMMA 8. *The number of messages sent by the protocol in any time interval where k_{new} new tokens are added to the network is $O(n^2m + k_{\text{new}} \cdot n)$.*

Proof. Each message is a TOKEN, TOKEN_LEFT, ACK_TOKEN, or ACK_TOKEN_LEFT message. By property ($\mathcal{P}2$), the number of TOKEN messages sent in the time interval is $O(n^2m + k_{\text{new}} \cdot n)$. The number of TOKEN_LEFT messages sent is at most the sum of the counters *token_left_pending* at the beginning of the interval, plus the number of TOKEN messages sent in the time interval. By Corollary 4 and the above, this sums to $O(2nm + n^2m + k_{\text{new}} \cdot n)$.

The number of ACK_TOKEN and ACK_TOKEN_LEFT messages sent, in the time interval under consideration, is at most the number of TOKEN and TOKEN_LEFT messages sent in this time interval, plus the number of messages that were in transit when the time interval started. Since the capacity of each link is constant the number of messages in transit at any given time is $O(m)$, and we have a bound on the number of ACK messages (of both types) of $O(n^2m + k_{\text{new}} \cdot n + 2mn + m)$. □

LEMMA 9. *The message complexity of the protocol is $O(n^2m)$ messages.*

Proof. For every time interval $(t_{i-1}, t_i]$ the receiver receives $2\cdot\mathcal{C}+1$ tokens during the interval. Since the capacity of the network is $\mathcal{C}$ tokens, at most $3\cdot\mathcal{C}+1$ tokens are sent by the sender in $(t_{i-1}, t_i]$. As $\mathcal{C} = O(nm)$, the lemma follows from Lemma 8. □

Since messages have size at most the size of the data-item, we establish the following theorem.

THEOREM 10 (communication complexity). *The communication complexity of the above protocol is* $O(n^2mD)$ *bits.*

THEOREM 11 (space complexity). *The space required at each processor is* $O(D + \log n)$ *bits per incident link.*

Proof. The list of variables used by each processor (per incident link) is given in section 3.2. By Corollary 4, the value of each of the counters *message_tokens*, *nil_tokens*, and *tokens_left_pending* is at most n, and hence requires only $O(\log n)$ bits. By Lemma 3, the same is true for the counter *bound*. In addition, there is a constant number of single-bit flags, per link. Finally, the processors store a single variable, *current_message*, of size D bits. □

5. Conclusions. In this work, we give an end-to-end communication protocol that has polynomial communication complexity and at the same time logarithmic space complexity. Thus we show that it is possible to attain polynomial communication complexity and sublinear space complexity at the same time. It remains, however, open, whether *constant* space complexity (more precisely, space complexity $O(D)$) allows for polynomial, or even merely bounded, communication complexity protocols in the above setting.

Appendix A. Proofs of technical lemmas.

Proof of Lemma 1. Upon initialization, the invariant holds. This is because the variable *bound*[e] is initialized to 1, the counters *message_tokens*[e], *nil_tokens*[e], and *tokens_left_pending*[e] are initialized to 0, and no message is in transit in the network. By induction on the events that change any of the values participating in the invariant, we show that the invariant holds for any time t. There are six types of events to be considered: send and receive events of TOKEN messages from u to v on e; send and receive events of TOKEN_LEFT messages from v to u on e; send events of TOKEN messages from v, when the token sent is accounted for in the counters of e in v; and receive events of a TOKEN at v, when this arrival causes a token accounted for at e to become a nil-token.

If both v and u are *not* the receiver, then the following describes the effects of any of the events:

- Send event of a TOKEN message from u to v on e: *bound*$[e]_u$ is incremented by 1, but so is $tokens_{u\to v}$.
- Receive event of a TOKEN message at v on link e: the sum *message_tokens*$[e]_v$ + *nil_tokens*$[e]_v$ is incremented by 1, but $tokens_{u\to v}$ is decremented by 1.
- Send event of a TOKEN_LEFT message from v to u on e: *tokens_left_pending*$[e]_v$ is decremented by 1, but $tokens\_left_{v\to u}$ is incremented by 1.
- Receive event of a TOKEN_LEFT message at u from v, on e: *bound*$[e]_u$ is decremented by 1, but so is $tokens\_left_{v\to u}$.
- A send event of a TOKEN message from u on some link e', when the token sent was accounted for at the counters of e: *message_tokens*[e] and *nil_tokens*[e] are decremented by 1, but *tokens_left_pending*[e] is incremented by 1.
- A receive event of a TOKEN message at v (on some link) may cause a token accounted for at the counters of e to become nil: *message_tokens*[e] is decremented by 1, but *nil_tokens*[e] is incremented by 1.

If v is the receiver:

- Send event of a TOKEN message from u to v on e: *bound*$[e]_u$ is incremented by 1, but so is $tokens_{u\to v}$.
- Receive event of a TOKEN message at v on e: $tokens_{u\to v}$ is decremented by 1 but *tokens_left_pending*[e] is incremented by 1.

- Send event of a TOKEN_LEFT message from v to u on e: $tokens\_left\_pending[e]_v$ is decremented by 1, but $tokens\_left_{v\to u}$ is incremented by 1.
- Receive event of a TOKEN_LEFT message at u from v, on e: $bound[e]_u$ is decremented by 1, but so is $tokens\_left_{v\to u}$.
- A send event of a TOKEN message from u on some link e', when the token sent was accounted for at the counters of e: one of the counters $message\_tokens[e]$ and $nil\_tokens[e]$ is decremented by 1, but $tokens\_left\_pending[e]$ is incremented by 1. (However, note that this event cannot happen, since u never receives any token on e.)
- A receive event of a TOKEN message at v (on some link) never changes the values of $message\_tokens[e]$ or $nil\_tokens[e]$.

If u is the receiver:

- A send event of a TOKEN message from u on e cannot happen, by the code.
- Receive event of a TOKEN message at v on link e: the sum $message\_tokens[e]_v + nil\_tokens[e]_v$ is incremented by 1, but $tokens_{u\to v}$ is decremented by 1 (note, however, that such an event cannot happen as u does not send TOKEN messages).
- Send event of a TOKEN_LEFT message from v to u on e: $tokens\_left\_pending[e]_v$ is decremented by 1, but $tokens\_left_{v\to u}$ is incremented by 1 (note that this event cannot happen too).
- Receive event of a TOKEN_LEFT message at u from v, on e: $bound[e]_u$ is decremented by 1, but so is $tokens\_left_{v\to u}$ (note that this event cannot happen).
- A send event of a TOKEN message accounted for at u cannot happen, by the code.
- A receive event of a TOKEN message at v (on some link) may cause a token accounted for at the counters of e to become nil: $message\_tokens[e]$ is decremented by 1, but $nil\_tokens[e]$ is incremented by 1. □

Proof of Lemma 2. Let t be the time just before the token is sent from u, and t' the time just before it is received at v. Because the sum of $message\_tokens$ and nil_tokens can increment only when tokens arrive on the link, and because the links are FIFO, we have:

$$\begin{aligned} &message\_tokens[e]_v^{t'} + nil\_tokens[e]_v^{t'} \\ &\le message\_tokens[e]_v^{t} + nil\_tokens[e]_v^{t} + tokens_{u\to v}^{t}. \end{aligned}$$

By Lemma 1,

$$message\_tokens[e]_v^t + nil\_tokens[e]_v^t + tokens_{u\to v}^t + 1 \le bound[e]_u^t \ .$$

Thus,

$$message\_tokens[e]_v^{t'} + nil\_tokens[e]_v^{t'} + 1 \le bound[e]_u^t.$$

By the code, $i > bound[e]_u^t$ and $j = message\_tokens[e]_v^{t'} + nil\_tokens[e]_v^{t'} + 1$, hence $i > j$. □

Proof of Lemma 3. We prove the claim by contradiction. Assume $bound[e]_u^t > n$ for some t, u, and $e = (u, v)$. Then a token must have been sent over e from u to v when $bound[e]_u \ge n$. By the code, this can happen only if, for some $e' \in E \cup \{I\}$, $message\_tokens[e']_u + nil\_tokens[e']_u > n$. However, for any e', the value of $message\_tokens[e']_u + nil\_tokens[e']_u$ is initialized to 0. Therefore, consider the

first time that, for some e' and u, $message\_tokens[e']_u + nil\_tokens[e']_u > n$. For the "virtual link" I at the sender this cannot happen by the code. For every $e' = (u, v') \in E$, the value of $message\_tokens[e']_u + nil\_tokens[e']_u$ increases only when a token is received at u over e'. Consider the token upon whose receipt the value of $message\_tokens[e']_u + nil\_tokens[e']_u$ became strictly larger than n, and denote by e'' the link incident to v whose counters accounted for this token before it was sent from v. By Lemma 2, when the token was sent from v, the value of $message\_tokens[e'']_v + nil\_tokens[e'']_v$ was already strictly greater than n, contradicting the fact that we are considering the first such event. □

Proof of Corollary 4. For $e = I$, the claim follows from the code. For any $e \in E$, by Lemmas 1 and 3,

$$message\_tokens[e]^t_v + nil\_tokens[e]^t_v + tokens^t_{u \to v} \leq bound[e]^t_u - 1 \leq n - 1 \ ,$$

and

$$tokens\_left\_pending[e]^t_v \leq bound[e]^t_u - 1 \leq n - 1 \ . \qquad \square$$

REFERENCES

[AAF+90] Y. Afek, H. Attiya, A. Fekete, M. J. Fischer, N. Lynch, Y. Mansour, D. Wang, and L. D. Zuck, *Reliable communication over unreliable channel*, J. Assoc. Comput. Mach., 40 (1993), pp. 1087–1107.

[AAG+97] Y. Afek, B. Awerbuch, E. Gafni, Y. Mansour, A. Rosén, and N. Shavit, *Slide – the key to polynomial end-to-end communication*, J. Algorithms, 22 (1997), pp. 158–186.

[AAG87] Y. Afek, B. Awerbuch, and E. Gafni, *Applying static network protocols to dynamic networks*, in Proc. 28th IEEE Ann. Symp. on Foundations of Computer Science, 1987, pp. 358–370.

[AAM89] Y. Afek, B. Awerbuch, and H. Moriel, *A Complexity Preserving Reset Procedure*, Technical Report MIT/LCS/TM-389, MIT, Cambridge, MA, May 1989.

[AE86] B. Awerbuch and S. Even, *Reliable broadcast protocols in unreliable networks*, Networks, 16 (1986), pp. 381–396.

[AG88] Y. Afek and E. Gafni, *End-to-end communication in unreliable networks*, in Proc. 7th ACM Symp. on Principles of Distributed Computing, August 1988, pp. 131–148.

[AG91] Y. Afek and E. Gafni, *Bootstrap network resynchronization: An efficient technique for end-to-end communication*, in Proc. 10th Ann. ACM Symp. on Principles of Distributed Computing, August 1991, pp. 295–307.

[AG90] A. Arora and M. Gouda, *Distributed reset*, in Proc. 10th Conf. on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Comp. Sci. 472, Springer-Verlag, New York, 1990, pp. 316–331.

[AGH90] B. Awerbuch, O. Goldreich, and A. Herzberg, *A quantitative approach to dynamic networks*, in Proc. 9th ACM Symp. on Principles of Distributed Computing, August 1990, pp. 189–204.

[AGR92] Y. Afek, E. Gafni, and A. Rosén, *The slide mechanism with applications in dynamic networks*, in Proc. 11th ACM Symp. on Principles of Distributed Computing, August 1992, pp. 35–46.

[AKM+93] A. Awerbuch, S. Kutten, Y. Mansour, B. Patt-Shamir, and G. Varghese, *Time optimal self-stabilizing synchronization*, in Proc. 25th ACM Symp. on the Theory of Computing , 1992, pp. 652–661.

[AKY90] Y. Afek, S. Kutten, and M. Yung, *Memory-efficient self-stabilization on general networks*, in Proc. 4th Workshop on Distributed Algorithms, Italy, 1990, Lecture Notes in Comp. Sci. 486, Springer-Verlag, New York, 1990, pp. 15–28.

[AL93] B. Awerbuch and T. Leighton, *A simple local-control approximation algorithm for multi-commodity flow*, in Proc. 34th IEEE Symp. on Foundations of Computer Science, 1993, pp. 459–469.

[AL94] B. Awerbuch and T. Leighton, *Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks*, in Proc. 26th ACM Symp. on the Theory of Computing, 1994, pp. 487–496.

[AM88] B. Awerbuch and Y. Mansour, *An efficient topology update protocol for dynamic networks*, Unpublished manuscript, January 1988.

[AMS89] B. Awerbuch, Y. Mansour, and N. Shavit, *Polynomial end to end communication*, in Proc. 30th IEEE Ann. Symp. on Foundations of Computer Science, October 1989, pp. 358–363.

[AO94] B. Awerbuch and R. Ostrovsky, *Memory-efficient and self-stabilizing network RESET*, in Proc. 13th ACM Symp. on Principles of Distributed Computing, August 1994, pp. 254–263.

[APV91] B. Awerbuch, B. Patt-Shamir, and G. Varghese, *Self-stabilization by local checking and correction*, in Proc. 32nd IEEE Ann. Symp. on Foundations of Computer Science, October 1991, pp. 268–277.

[AS88] B. Awerbuch and M. Sipser, *Dynamic networks are as fast as static networks*, in Proc. 29th IEEE Ann. Symp. on Foundations of Computer Science, October 1988, pp. 206–220.

[AV91] B. Awerbuch and G. Varghese, *Distributed program checking: A paradigm for building self-stabilizing distributed protocols*, in Proc. 32nd IEEE Ann. Symp. on Foundations of Computer Science, October 1991, pp. 258–267.

[BS88] A. E. Baratz and A. Segall, *Reliable link initialization procedures*, IEEE Trans. Comm., 36 (1988), pp. 144–153; also in IFIP 3rd Workshop on Protocol Specification, Testing and Verification, III.

[Dij74] E. Dijkstra, *Self stabilization in spite of distributed control*, Comm. ACM, 17 (1974), pp. 643–644.

[DF88] E. W. Dijkstra and W. H. J. Feijin, *A Method of Programming*, Addison-Wesley, Reading, MA, 1988.

[DW93] S. Dolev and J. Welch, *Crash resilient communication in dynamic networks*, in Proc. 7th International Workshop on Distributed Algorithms, Lecture Notes in Comp. Sci. 725, Springer-Verlag, New York, 1993, pp. 129–144.

[Fin79] S. Finn, *Resynch procedures and a fail-safe network protocol*, IEEE Trans. Comm., COM-27 (1979), pp. 840–845.

[IL94] G. Itkis and L. Levin, *Fast and lean self-stabilizing asynchronous protocols*, in Proc. 35th IEEE Ann. Symp. on Foundations of Computer Science, November 1994, pp. 226–239.

[KP90] S. Katz and K. Perry, *Self-stabilizing extensions for message-passing systems*, in Proc. 9th ACM Symp. on Principles of Distributed Computing, August 1990, pp. 91–101.

[MOOY92] A. Mayer, Y. Ofek, R. Ostrovsky, and M. Yung, *Self-stabilizing symmetry breaking in constant-space*, in Proc. 24th ACM Symp. on the Theory of Computing, 1992, pp. 667–678.

[Vis83] U. Vishkin, *A distributed orientation algorithm*, IEEE Trans. Inform. Theory, 29 (1983), pp. 624–629.

SIAM J. COMPUT.
Vol. 27, No. 6, pp. 1550–1563, December 1998
© 1998 Society for Industrial and Applied Mathematics
003

LOWER BOUNDS FOR RANDOMIZED MUTUAL EXCLUSION*

EYAL KUSHILEVITZ†, YISHAY MANSOUR‡, MICHAEL O. RABIN§, AND DAVID ZUCKERMAN¶

Abstract. We establish, for the first time, lower bounds for *randomized* mutual exclusion algorithms (with a read-modify-write operation). Our main result is that a constant-size shared variable cannot guarantee strong fairness, even if randomization is allowed. In fact, we prove a lower bound of $\Omega(\log \log n)$ bits on the size of the shared variable, which is also tight.

We investigate weaker fairness conditions and derive tight (upper and lower) bounds for them as well. Surprisingly, it turns out that slightly weakening the fairness condition results in an exponential reduction in the size of the required shared variable. Our lower bounds rely on an analysis of Markov chains that may be of interest on its own and may have applications elsewhere.

Key words. mutual exclusion, randomized distributed algorithms, Markov chains, lower bounds

AMS subject classifications. 68Q22, 68M99, 60J10

PII. S009753979426513X

1. Introduction.

Randomization has played an important role in the design and understanding of distributed algorithms. It is a natural tool which is usually used in order to break symmetry between identical processes in a distributed system. Beyond its natural role in symmetry breaking, randomization often increases the computation power (e.g., [LR81, Ben83]), significantly decreases computational costs (e.g., [Bra85, FM88]), and helps in simplifying algorithms.

For many applications in distributed environments, there is a provable gap between the power of randomized algorithms and that of their deterministic counterparts. The most renowned example is achieving Byzantine agreement with a linear number of faults; while any deterministic algorithm requires at least a linear number of rounds [FL82], there is a randomized algorithm that performs the same task in a constant number of rounds [FM88]. Another important example is that of reaching a consensus in an asynchronous distributed system with faults: this is impossible with deterministic protocols, even if the faults are restricted to a single fail-stop fault [FLP85], but is possible with the use of randomized protocols (see [CIL87]).

The gap between the performances of randomized and deterministic algorithms exists also for the *mutual exclusion* problem. The complexity measure here is the

* Received by the editors February 1, 1994; accepted for publication (in revised form) September 5, 1996; published electronically June 3, 1998. An early version of this paper appeared in *Proc. of the 25th ACM Symp. on Theory of Computing*, 1993, pp. 154–163. The research of Eyal Kushilevitz and Michael Rabin was supported by research contracts ONR-N0001491-J-1981 and NSF-CCR-90-07677 at Harvard University.

http://www.siam.org/journals/sicomp/27-6/26513.html

† Dept. of Computer Science, Technion, Haifa 32000, Israel (eyalk@cs.technion.ac.il, http://www.cs.technion.ac.il/∼eyalk). Part of this research was done while the author was at Aiken Computation Lab., Harvard University, Cambridge, MA 02138.

‡ Computer Science Dept., Tel-Aviv University and IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 (mansour@math.tau.ac.il).

§ Aiken Computation Lab., Harvard University, Cambridge, MA 02138-2901 and Institute of Mathematics, Hebrew University of Jerusalem, Givat Ram, Jerusalem, Israel (rabin@das.harvard.edu).

¶ Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712 (diz@cs.utexas.edu). This research was done while the author was at the Laboratory for Computer Science, MIT, and supported by an NSF Postdoctoral Fellowship.

size of the shared variable.[1] Any deterministic algorithm requires an $\Omega(\log n)$-bit shared variable, in order to achieve mutual exclusion (with fairness) between n distinct processes, and this bound is tight [BFJ+82]. On the other hand, there is a randomized algorithm requiring only an $O(\log \log n)$-bit shared variable [Rab82, KR92].[2]

It remained an open problem whether the complexity of the randomized algorithm for the mutual exclusion problem could be further reduced, perhaps even to a constant number of bits. A constant-size shared variable is of special interest, since it implies that the size of the shared memory can be independent of the number of the processes using it. Our main contribution is a tight $\Omega(\log \log n)$ lower bound on the number of bits required to implement the shared variable. Other tight *upper* and *lower* bounds are given for mutual exclusion with weaker fairness properties.

Few lower bounds are known for randomized distributed algorithms. Many of these lower bounds are based on arguments that arise from the need for information to flow from one side of the network to the other side, or based on the symmetry between different processes [IR81]. Another type of argument for randomized lower bounds is the use of the min-max theorem [Yao77, AS91]. For randomized Byzantine agreement, when more than a third of the processes are faulty, a lower bound on the success rate is known [KY84, GY89].

Previous work and our results. In the following, we give a more detailed description of the mutual exclusion problem and a summary of our results together with previous results related to our work.

The setting of the *mutual exclusion* problem is as follows. There are n processes that from time to time need to execute a critical section in which exactly one is allowed to employ some shared resource. The processes can coordinate their activities through a shared *read-modify-write* variable (i.e., reading and rewriting the shared variable is an atomic action). The sequence of accesses to the shared variable is determined by a scheduler.

The mutual exclusion problem is a classical problem in distributed computing. It was first suggested by Dijkstra [Dij65], who solved the problem using a (one-bit) semaphore. While the semaphore does solve the problem and guarantees *deadlock freedom*, it does not guarantee any *fairness* among the processes competing for the critical section; a process that is waiting for the critical section may wait forever. Since then, numerous solutions have been proposed for the mutual exclusion problem. All these solutions guarantee *deadlock freedom*, together with some notion of *fairness*.

An important parameter for evaluating the complexity of a mutual exclusion algorithm is the size of the shared variable that is used. As mentioned above, to guarantee only *deadlock-freeness*, a one-bit semaphore is sufficient [Dij65]. Burns et al. [BFJ+82] define the *bounded-waiting* property as a fairness criterion. Roughly speaking, this property guarantees that between the first time that a process accesses the shared variable in order to try to enter the critical section and the time it actually enters the critical section, each of the other processes may enter the critical section at most once. They proved that if *deterministic* algorithms are used, then an $\Omega(\log n)$-

[1] Throughout this work the size of the shared variable is measured in terms of the number of *bits* of the shared variable rather then the number of *values* (as is done in some of the papers in the literature). Clearly, the number of bits is logarithmic in the number of values, hence a constant factor in the number of bits translates to a polynomial factor in the number of values. Still, the number of bits is a very natural measure for the size of the variables.

[2] The first solution to this problem was given in [Rab82]. A flaw in this solution was pointed out by [Sai92]. A new solution, based on ideas of [Rab82], was given in [KR92].

bit shared variable is required for achieving bounded-waiting and that this number of bits is also sufficient.

Rabin [Rab82] suggested the use of *randomized* algorithms for mutual exclusion and defined the notion of fairness for such algorithms. Roughly speaking, the fairness of a randomized mutual exclusion algorithm measures the probability that a process enters the critical section at a given time, as a function of the number of processes concurrently competing for the critical section. Specifically, Rabin was interested in the following fairness property, which we refer to as *linear-fairness*: if a process participates in a "round"[3] together with m processes, it has probability $\Omega(1/m)$ of entering the critical section in the next round. This property can be considered as a probabilistic analogue of the bounded-waiting property. Randomized algorithms having the linear-fairness property that use $O(\log\log n)$-bit shared variable are presented in [Rab82, KR92]. This is in contrast to the $\Theta(\log n)$-bit shared variable required by deterministic algorithms.

Proving the correctness of such randomized distributed protocols involves many delicate issues. Saias [Sai92] developed a general methodology to prove the correctness of a randomized distributed protocol. The main difficulty of such proofs is the need to deal with two separate sources of nondeterminism: the randomness that the protocol generates and the decisions of the adversary. The key idea in his methodology is that these two ingredients should be made independent. Using his systematic methodology, Saias [Sai92] uncovered the flaw in [Rab82].

No *lower bounds* for randomized mutual exclusion were known. In fact, in light of the results mentioned earlier, it may seem plausible that a constant-size shared variable is sufficient for mutual exclusion with linear fairness. More than that, it was shown [Rab82, KR92] that a constant-size shared variable may be powerful; it suffices for guaranteeing that each of the competing processes will have $\Omega(1/n)$ probability of entering the critical section. However, this is *independent* of m and hence is much weaker. In this paper, we prove a tight $\Omega(\log\log n)$-bit lower bound on the size of the shared variable that is needed for achieving mutual exclusion with linear-fairness. Thus, in particular, a constant-size shared variable cannot guarantee linear-fairness.

We define a slightly weaker fairness property that we term *polynomial-fairness*: if a process participates in a round together with m processes, it has a probability of $\Omega(1/m^{1+\varepsilon})$ to enter the critical section in the next round (where $\varepsilon > 0$ is a constant). Surprisingly, we show that for every $\varepsilon > 0$ an $O(\log\log\log n)$-bit shared variable is sufficient to achieve polynomial fairness, and that an $\Omega(\log\log\log n)$-bit shared variable is necessary. Hence, this slight weakening of the fairness property results in an exponential reduction in the size of the required shared variable. Finally, we show that with a constant-size shared variable it is possible to guarantee an $\Omega(1/2^m)$ probability of entering the critical section.

For our lower-bound proofs, we study general *Markov chains*, i.e., those which are represented by a $k \times k$ real nonnegative matrix where the sum of elements in each row is 1. We call a Markov chain *live* if in each of its first n steps it has a "considerable probability" of visiting a new state (i.e., a state that was not visited in each of the previous steps). We relate the fairness of mutual exclusion algorithms to the liveness of Markov chains (where the different notions of fairness correspond to different interpretations of "considerable probability"). We obtain our bounds for mutual exclusion by proving bounds on k, the number of states of the Markov chains (as a function of n). We believe that our bounds and technique may be found useful

[3] A round is the time between two consecutive closings of the critical section.

for other applications.

The paper is organized as follows. In section 2 we formally define the mutual exclusion problem and Markov chains. In sections 3 and 4 we prove the lower bounds for linear/polynomial-fairness (respectively). Finally, in section 5 we show the upper bounds.

2. Preliminaries.

2.1. Mutual exclusion. In this section we define the properties required from a randomized mutual exclusion algorithm. Let $P_1, \ldots, P_n$ be the n processes in the system. The processes coordinate their activities by using a shared read-modify-write variable v. (In addition, each process P_i has unbounded local memory.) While it is convenient to assume that all the processes run the same program, our results do not depend on this assumption. During the computation, each process P_i is in one of four possible phases: *TRYING phase*, in which it attempts to enter the critical section, *CS* (*critical section*) *phase*, in which it executes the critical section, *EXIT phase*, in which it leaves the critical section, or *REMAINDER phase*, in which it does other local computations.

At any given time the adversary scheduler[4] can observe the *external behavior* of the processes (i.e., which of the four phases each process currently executes) and use this information (together with its information on the past behavior of the processes) to determine which process will be the next to access the shared variable. (It is also assumed that the adversary knows the algorithm used by the processes, including the initial state of each process.) The adversary scheduler *cannot* observe the content of the shared variable nor the content of any local variable.[5] More formally, let a *run* be a (finite or infinite) sequence $(i_1, x_1), \ldots, (i_k, x_k), \ldots$, where x_j indicates which phase process P_{i_j} started or whether it accessed the shared variable. A run is called *proper* if the subsequence of phases corresponding to every process P_i is of the form REMAINDER, TRYING, CS, EXIT, REMAINDER, ...

A *scheduler* is a (probabilistic) function that on a finite run σ gives the identity of the next process to access the shared variable. It should satisfy the following property.

Scheduler-liveness: For each time t and any process P_j not in REMAINDER phase at time t, there exists a time $t' > t$ in which P_j makes a move.

Next, we discuss the correctness conditions of a randomized mutual exclusion algorithm. While the first three conditions are rather standard, the fairness definition is the one unique to the randomized solutions. These correctness conditions may be generalized in various ways without affecting the results. We discuss possible extensions of the conditions throughout the paper.

Mutual exclusion: At any time t there is at most one process in the CS phase. If there is a process in the CS phase then we say that the critical section is *closed*; otherwise, it is *open*.

Fault-freeness: If a process P_j moves from TRYING phase to the CS phase then eventually P_j moves from the CS phase to EXIT phase and from EXIT phase to REMAINDER phase. (For example, a protocol in which a process after entering the critical section gets into an infinite loop violates this condition.)

Deadlock-freeness: If the critical section is *open* and there is a process in TRYING phase, then eventually some process enters the CS phase.[6]

[4] We assume here the same adversary scheduler and the same correctness conditions as in [KR92].

[5] Note that since we are interested in this work in proving *lower* bounds, this assumption makes our job more complicated.

[6] The definition can be weakened to require that this will hold with probability 1, and all the

f-fairness: Let C_i be the time at which the ith closing of the critical section occurs. Let S_i be the set of processes in TRYING phase that were scheduled to access the shared variable between time C_{i-1} and C_i, excluding the process that entered the critical section at C_i.

1. If $S_i \neq \emptyset$ then one of the processes in S_i enters the critical section at C_{i+1}.
2. For every process $P_j \in S_i$, the probability that P_j enters at time C_{i+1} is at least $1/f(t)$, where $t = |S_i|$.[7]

In particular, for a constant c, we refer to $c \cdot t$-fairness as *linear*-fairness, and to t^c-fairness as *polynomial*-fairness.[8]

In the definition of f-fairness, we require that a process that tries to enter at time C_i will have a "good" chance to enter at the *next* time, i.e., C_{i+1}. At first sight, it seems more natural to require that a process that arrives between time C_{i-1} and C_i will have a good probability to enter at C_i, as defined in [Rab82]. However, as pointed out by [Sai92], such a statement is circular since the definition of the event C_i depends on whether the process enters the critical section or not, and it seems that there is no "acceptable" way to get around this problem. We follow here the solution suggested by [KR92], which requires the "good" chance to be only in the next time step.

2.2. Markov chains. Let $S = \{s_1, s_2, \ldots, s_k\}$ be a set of k states. A *Markov chain* is a real, nonnegative, $k \times k$ matrix $(Q_{i,j})$ with the property that the sum of elements in each row equals 1. It can be used to generate sequences of elements of S in the following way. Start in the initial state, say, s_1. At each step, if the last element in the sequence is s_i, move to state s_j with probability $Q_{i,j}$ (i.e., the probability of moving into state s_j depends only on the last state s_i and not on the whole history).

We say that a Markov chain $(Q_{i,j})$ is $(n, f(t))$-*live* if for every $1 \leq t \leq n$ the probability that the sequence generated by the above process visits in the tth step a state that was not visited during the first $t-1$ steps is at least $1/f(t)$.

It is sometimes convenient to think about the Markov chain as a complete directed graph on k nodes. Every edge $i \to j$ has a value $Q_{i,j}$ which is the probability of visiting node j in the next step when being in node i. The sequence of states, in this case, is usually called a *walk*.

3. Lower bound: Linear-fairness. In this section we describe the lower bound for the case of *linear-fairness*; that is, where the probability of each process entering the critical section is required to be inversely proportional to the number of processes trying to enter the critical section. To do so, we describe a strategy for the adversary scheduler, given a mutual exclusion algorithm $\mathcal{A}$, to plan a schedule in which the probability that a certain process (that the adversary wish to discriminate against) enters the critical section in a given round is smaller than what is required.

---

results of the paper will remain valid.

[7] The probability space is defined on prefixes of runs; therefore, the space is finite. Formally, the above requirement says that for any prefix of a run up to C_i, σ, which have a nonzero probability, the probability that P_j enters at time C_{i+1} given σ is at least $1/f(t)$. Later, when we will refer to an event as "happened in the past" we will mean that it is satisfied by σ.

[8] It is possible to relax the definition of fairness and to allow each party that was scheduled to access the shared variable between times C_{i-1} and C_i and that has not entered the critical section at times $C_i, C_{i+1}, \ldots, C_{i+d-1}$, for some parameter d, to compete on entering at time C_{i+d} with probability of success at least $1/f(t)$. The results and the proofs (with few minor changes) hold for such a definition as well.

1. Schedule P_n until it enters the critical section.
2. Schedule $P_1, \ldots, P_s, \ldots, P_t$ (each is scheduled once).
3. Schedule P_n until it exists the critical section.
4. Schedule $P_1, \ldots, P_s$ for M steps (in a round-robin).
5. Schedule $P_1, \ldots, P_n$ (in a round-robin).

FIG. 1. *Strategy for the adversary* (*assuming the existence of* s *and* t *as in Lemma* 1).

The schedule starts by scheduling the process P_n to access the shared variable until this process enters the CS phase; i.e., the critical section is *closed* (Figure 1, step 1). The deadlock-freeness property guarantees that this will eventually happen. Denote by d the number of steps taken by P_n before entering the CS phase, and by $v[n]$ the value that P_n wrote into the shared variable at this time. Then, the adversary schedules each of the processes P_1 to P_t (in order) to perform a single read-modify-write operation on the shared variable, where t is a parameter (Figure 1, step 2). We denote by $v[i]$ the value written by P_i into the shared variable. The proof of the lower bound has two parts. We first show that if t is *good* (in a sense that will be defined later) then the adversary can discriminate against P_t; namely, with high probability, process P_t will not enter the critical section (although scheduled to access the shared variable). Later, we show that if the shared variable is "too small" then a *good* t must exist. To introduce the idea of the proof we first assume that t satisfies an even stronger property, as formalized in the next lemma.

LEMMA 1. *Assume that there exists a specific* s, $1 < s < t$, *such that the probability that the value* $v[t]$ *written by* P_t *into the shared variable equals the value* $v[s]$ *written by* P_s *is at least* $1-\delta$. *Then there exists an extension of the above schedule such that the probability that* P_t *enters at either* C_1 *or* C_2 *is at most* $\delta+\varepsilon$, *where* ε *is arbitrarily small.*

Proof. Assume that $v[t] = v[s]$. That is, P_t wrote into the shared variable the same value as P_s (this happens with probability at least $1-\delta$). The adversary extends the schedule by first scheduling P_n to access the shared variable until it moves to the REMAINDER phase and the critical section is open (Figure 1, step 3). This is guaranteed by the fault-freeness property. Then (Figure 1, step 4), the adversary continues by scheduling only P_1 to P_s (say, by a round-robin). Observe that the only way that a process P_i can note that another process P_j was scheduled before it is if P_j changed the value of the shared variable. Hence, if indeed $v[t] = v[s]$, then in this case processes P_1 to P_s must operate as if P_{s+1} to P_t were not scheduled. This implies (by the deadlock-freeness property and the first part of the fairness property) that eventually some process P_i enters at C_1, and some other process P_j at C_2, where $1 \le i, j \le s$. More precisely, there exists a large enough M such that if $P_1, \ldots, P_s$ are scheduled to take M steps, then with probability at least $1-\varepsilon$, two of these processes enter at time C_1 and time C_2 (if this does not happen during the M steps this could be either because P_t did *not* write the same value as P_s or because none of $P_1, \ldots, P_s$ entered the critical section). Hence, the probability that P_t enters at either C_1 or C_2 is bounded by the probability that it did not write the same value as P_s plus the probability that M steps were not enough for $P_1, \ldots, P_s$, which is at most $\delta+\varepsilon$. ∎

The problem with the above lemma is that the adversary needs to know some fixed s, such that the probability that P_t writes to the shared variable the same value that was written by P_s, is "high." The next lemma shows that the same bound holds even when s is not fixed. First we define the notion of *good* t.

1. Schedule P_n until it enters the critical section.
2. Schedule $P_1, \ldots, P_t$ (each is scheduled once).
3. Schedule P_n until it exists the critical section.
4. For $i = 1, \ldots, t-1$
   Schedule $P_1, \ldots, P_i$ for M_i steps (in a round-robin).
5. Schedule $P_1, \ldots, P_n$ (in a round-robin).

FIG. 2. *Strategy for the adversary* (*assuming the existence of* t *which is* δ*-good*).

DEFINITION 1. *We say that* t *is* δ-good[9] *if the probability that the value* $v[t]$ *written by* P_t *into the shared variable equals one of* $v[1], \ldots, v[t-1]$ (*the values written by* $P_1, \ldots, P_{t-1}$, *respectively*) *is at least* $1 - \delta$.

LEMMA 2. *Given that* t *is* δ*-good, there exists an extension of the above schedule such that the probability that* P_t *enters at either* C_1 *or* C_2 *is at most* $\delta + \varepsilon$, *where* ε *is arbitrarily small.*

Proof. As t is δ-good, the adversary, who knows the algorithm used, knows that with high probability there exists an s ($1 \leq s < t$) such that $v[t]$ equals $v[s]$. However, the adversary has to overcome the fact that he does not know the value of s. He will do so by trying all the possible values of s. The problem is that trying one value of s, say $s = 9$, influences other values of s, say $s = 3$, since the processes may notice that $s > 3$. The first idea is to try $s = 1, 2, \ldots, t-1$ in this order. This guarantees that before trying $s = i$, the only processes scheduled are $P_1, \ldots, P_{i-1}$ (this is done by modifying step 4; see Figure 2).

Essentially, when the adversary checks whether $s = i$, it does the same thing as in the proof of Lemma 1 above. Namely, it schedules only $P_1, \ldots, P_i$. If, indeed, $s = i$, then the adversary is guaranteed that if it schedules only $P_1, \ldots, P_i$, eventually one of them would enter at C_1 and one at C_2 (it might be the same process in both cases). If the scheduler detects that $s \neq i$, then it continues to $i+1$. We are guaranteed that, with probability $1 - \delta$, there exists such an i.

We need to show how the adversary can check whether $s = i$ or $s \neq i$. If the adversary has a bound on the number of steps until the processes $P_1, \ldots, P_i$ would let one in C_1 and another in C_2, say M_i steps, it would schedule them this many steps. If no process would enter at either C_1 or C_2, then the scheduler is guaranteed that $s \neq i$. As in the proof of Lemma 1 it may be the case that such a bound M_i does not exist. For this reason it would compute the value of M_i such that if $s = i$ the probability that one of $P_1, \ldots, P_i$ enters at C_1 and C_2 during at most M_i steps (in a round-robin schedule) is at least $1 - \varepsilon$.

The probability that the scheduler misses the right value of s is ε (note that we do not care about the other cases). In addition, we assumed that there exists such an s, with probability $1 - \delta$. Therefore, the probability that P_t enters is at most $\delta + \varepsilon$. This is since this probability is bounded by the probability that there is no such s (bounded by ε), plus the probability that, given that there is such an s, the scheduler misses it (bounded by δ). □

So far, we proved that if there is a t which is δ*-good* then the adversary can discriminate against P_t. We now prove that such a t must exist if the number of values is "too small." At this point it is convenient to define the Markov chain $Q(\mathcal{A}, d)$ corresponding to a mutual exclusion algorithm $\mathcal{A}$ and an integer d (where d

[9] The term "good" is from the adversary point of view.

will be taken as the number of times P_n was scheduled before entering the critical section; this parameter is known to the adversary). Note that d does depend on $\mathcal{A}$, but for each specific d we have a different Markov chain. Recall that we assume, at this point, that all the processes run the same program.

States: The states of the Markov chain correspond to the possible values of the shared variable. In addition there is a special initial state q_0 (i.e., if we have a k-bit shared variable, then the Markov chain has $2^k + 1$ states).

Transition probabilities: For $i, j \geq 1$, the entry $Q_{i,j}$ equals the probability that a process, when invoked for the first time (i.e., it is in its initial state) and reading the value i from the shared variable, writes the value j. This probability is defined by the algorithm $\mathcal{A}$. For the initial state and $i > 0$, we define $Q_{0,i}$ to be the probability that the process P_n, before closing the critical section, wrote the value $v[n] = i$ (this probability depends on d!). Also, $Q_{i,0} = 0$ for all i.

The idea is that the behavior of a process which is scheduled to read the shared variable for the first time depends only on the current value of the shared variable and does not depend on the whole history of values. This Markov property enables us to describe the process of writes as a Markov chain. The relation between this Markov chain and the schedule we are constructing is formalized by the following claim; later we concentrate on analyzing the Markov chain.

CLAIM 1. *Fix a schedule as above. Also, let $\mathcal{A}$, d, and $Q(\mathcal{A}, d)$ be as above. Then, for every sequence of values $V_n, V_1, \ldots, V_i$,*

$$\begin{aligned}\Pr[(s_0 = V_n) \wedge (s_1 = V_1) \wedge \ldots \wedge (s_i = V_i)] \\ = \Pr[(v[n] = V_n) \wedge (v[1] = V_1) \wedge \ldots \wedge (v[i] = V_i)],\end{aligned}$$

where $s_0, s_1, s_2, \ldots$ is the sequence of states visited by the Markov chain.

Proof. The proof follows by an easy induction from the definition of the Markov chain. □

It follows from the definitions that if every t is not $\frac{1}{ct}$-good, then the corresponding Markov chain is $(n, c \cdot t)$-live; hence, if the Markov chain is not (n, ct)-live, then there exists a t which is $\frac{1}{ct}$-good. (Recall that a Markov chain $(Q_{i,j})$ is $(n, f(t))$-*live* if for every $1 \leq t \leq n$ the probability that the tth step reaches a state that was not visited during the first $t - 1$ steps is at least $1/f(t)$.) The following lemma gives a bound on the number of states of *any* Markov chain (not only those constructed as above) with linear-liveness property.

LEMMA 3. *Let $c \geq 0$ be any constant. Let $(Q_{i,j})$ be any Markov chain on k states which is $(n, c \cdot t)$-live. Then, $k > \frac{1}{c} \ln n$.*

Proof. For every $1 \leq i \leq n$, let X_i be a random variable which takes the value 1 if the Markov chain visits a new state in its ith step, and 0 otherwise. Clearly, $\sum_{i=1}^{n} X_i$ is at most the number of states k, and hence also $E[\sum_{i=1}^{n} X_i] \leq k$. By linearity of expectation, $E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i]$. By the liveness of the Markov chain, for every i, $\text{Prob}[X_i = 1] \geq \frac{1}{ci}$. This implies that $E(X_i) \geq \frac{1}{ci}$. Combining all together, we get that $k \geq \sum_{i=1}^{n} \frac{1}{ci} > \frac{1}{c} \cdot \ln n$. □

THEOREM 4. *Every mutual exclusion algorithm $\mathcal{A}$ for n processes which guarantees $O(t)$-fairness requires a shared variable of $\Omega(\log \log n)$ bits.*

Proof. Consider the algorithm $\mathcal{A}$ and the corresponding Markov chain $Q(\mathcal{A}, d)$ and assume that the algorithm $\mathcal{A}$ guarantees ct-fairness. If the Markov chain $Q(\mathcal{A}, d)$ is not $(n - 1, 2ct)$-live, then there exists a t, $1 \leq t \leq n - 1$, which is $\frac{1}{2ct}$-good. By Lemma 2, there exists an extension of the basic schedule (in which P_n was scheduled d

times) such that the probability of P_t entering the critical section at either C_1 or C_2 is less than $\frac{1}{ct}$, contradicting the ct-fairness of the algorithm. Therefore, $Q(\mathcal{A}, d)$ must be $(n-1, 2ct)$-live. By Lemma 3, this implies that k, the number of states in this Markov chain, is at least $\frac{1}{2c}\ln(n-1)$. By the construction of the Markov chain, the number of bits in the shared variable used by $\mathcal{A}$ is $\log(k-1) \geq \log(\frac{1}{2c}\ln(n-1)-1) = \Omega(\log\log n)$, as claimed. □

Note that in the proof of Lemma 3 we do not use the fact that in each step we use the same transition matrix. In other words, the lemma holds even if we associate with every step i a different transition matrix $Q^{(i)}$. This implies that the lower bound of Theorem 4 still holds even if the processes are allowed to use different programs.

COROLLARY 5. *Every mutual exclusion algorithm for n processes which guarantees $O(t)$-fairness requires a shared variable of $\Omega(\log\log n)$ bits, even if each process runs a different program.*

In the above discussion, we assumed that the adversary knows which value of t is $\frac{1}{2ct}$-good. We can make this assumption because the adversary is given the algorithm $\mathcal{A}$, and it knows the number of steps d taken by P_n before entering the CS phase. Therefore, it can construct the above Markov chain. Based on this, the adversary can compute the probability of visiting a new state at any given step and hence find the value of t.

4. Lower bound: Nonlinear-fairness. In this section we extend the results from the case of linear-fairness to the case of polynomial (t^c, for $c > 1$) fairness. The proof goes along the same lines, except that the proof of Lemma 3 fails in this case, since $\sum_i 1/i^c = O(1)$, for $c > 1$. Thus, a different approach is required.

To simplify the proof, we assume that all the processes are identical (i.e., both code and initial state). At the end of the section we show that the proof can be extended to the case when the processes are not identical. Our goal now is to derive a lower bound for the number of states of (n, t^c)-live Markov chains.

Consider the k^2 values $Q_{i,j}$ of the Markov chain. We divide the proof into two cases according to the way these values are distributed in the interval $[0, 1]$. The easy case is when these values are "dense" in the interval. Lemma 6 below claims that in this case k must be "large." Then, we handle the more difficult case where there exists some "gap" in the interval $[0, 1]$ in which none of these k^2 values fall, and we show that in this case the Markov chain is not (n, t^c)-live.

LEMMA 6. *Let $\lambda > 1$ be a constant.*[10] *Let $(Q_{i,j})$ be a Markov chain over k states. If, for every $0 \leq \alpha \leq 1/2$ such that $\alpha^{\lambda k} \geq 1/n$, there exist i and j such that $Q_{i,j} \in (\alpha^{\lambda k}, \alpha]$, then $k = \Omega(\sqrt{\log\log n / \log\log\log n})$.*

Proof. Consider the sequence $\beta_\ell = 2^{-(\lambda k)^\ell}$ ($\ell = 0, 1, \ldots$). By the assumption, if $\beta_{\ell+1} \geq 1/n$ then the interval $(\beta_{\ell+1}, \beta_\ell]$ contains at least one of the values $Q_{i,j}$ (also note that these intervals are disjoint). Since there are at most k^2 such values, then $\beta_{k^2+1} < 1/n$; otherwise, not all the intervals contain a value $Q_{i,j}$. From this inequality we get that $k = \Omega(\sqrt{\log\log n / \log\log\log n})$. □

In the following we assume that there is such a gap; i.e., there exists an $\alpha \leq 1/2$ such that the interval $(\alpha^{\lambda k}, \alpha]$ contains no probability $Q_{i,j}$, and $\alpha^{\lambda k} \geq 1/n$. An edge $i \to j$ with probability $Q_{i,j} \leq \alpha^{\lambda k}$ is called α-light; otherwise, if $Q_{i,j} > \alpha$, it is called α-heavy. The assumption that there is a gap implies that every edge is either α-heavy or α-light. We consider a random walk of (a suitably chosen) length $t < n$. We show two main properties. The first is that the probability that in t steps of the Markov

[10] The value of λ depends on the value of the constant c.

chain some α-light edge is used is "small." The second is that the probability that we do not "cover" the graph induced by the α-heavy edges is "small." Before going into the details, we will make our choice of parameters, as follows:

$$t = \frac{\gamma k^2 c^2}{\alpha^{2k}} \qquad \text{and}$$
$$\lambda = \gamma' c \log c,$$

where γ and γ' are sufficiently large constants and k is the number of states. (Unfortunately, the best intuition that we can give for the choice of t and λ is that they make the proof go through.) We start by showing that the probability of traversing some α-light edge is negligible.

LEMMA 7. *The probability that any α-light edge is used in a walk of length t is less than $1/(2t^c)$.*

Proof. In each of the t steps, the walk can choose among at most $k-1$ α-light edges, each with probability at most $\alpha^{\lambda k}$. Therefore, the probability that any α-light edge is used is not more than $t \cdot k \cdot \alpha^{\lambda k}$. To see that this is less than $1/(2t^c)$, it is sufficient to show that $2t^{c+1}k\alpha^{\lambda k} < 1$. We now substitute the value of t into this inequality and we get that it is sufficient to prove $2\gamma^{c+1}k^{2c+3}c^{2c+2}\alpha^{(\lambda-2(c+1))k} < 1$. As $\alpha < 1/2$ it is enough that $2\gamma^{c+1}k^{2c+3}c^{2c+2} < 2^{(\lambda-2(c+1))k}$. This is satisfied as long as $(\lambda - 2(c+1))k > 1 + (c+1)\log\gamma + (2c+3)\log k + (2c+2)\log c$. Hence, choosing λ as above, with γ' sufficiently large, will satisfy the inequality, and the lemma follows. □

In the following we define what it means to cover a directed graph. Intuitively, a directed graph is covered by a walk if no new node can be reached.

DEFINITION 2. *A directed graph is* completely covered *by a walk W if each node that is reachable from the last node of the walk W has already been visited in W.*

Note that the above definition does *not* require that the walk visit *all* the nodes in the graph, just that there be no *new* nodes which can be reached from the last node. The next lemma gives a bound on the probability that we completely cover the graph induced by the α-heavy edges.

LEMMA 8. *Consider a Markov chain $(Q_{i,j})$ such that each transition probability is either α-heavy or α-light. The probability that after a walk W of t steps, which uses only α-heavy edges, the induced (directed) graph of α-heavy edges is not completely covered is less than $1/(2t^c)$.*

Proof. Recall that k is the number of states in the Markov chain. We divide the walk W into $b = \lceil t/k \rceil$ blocks of size k. Consider the location v of the walk at the beginning of a block. Either all the nodes reachable from v in the induced graph of α-heavy edges were already visited, or there is some node v', reachable from v, which was not visited yet. This implies that there is a (simple) path of length at most k from v to v' consisting of α-heavy edges. (There may be more than one such path; however, we cannot make any stronger assumption, e.g., the existence of an α-heavy edge connecting v to v'.) Therefore, the probability that the walk visits v' during the current block of steps is at least α^k.

By standard Chernoff bounds, the probability that the graph is not completely covered after t/k blocks is at most $e^{-\frac{t\alpha^k}{8k}}$. To see this, define a random variable X_i which is 1 if the graph is completely covered by the first $i-1$ blocks of the walk or if a new node is visited during the ith block of the walk. Otherwise, $X_i = 0$. By the above, the probability that X_i is 1 is at least $p = \alpha^k$. Let S be the sum of these random variables. That is, $S \stackrel{\triangle}{=} \sum_{i=1}^{b} X_i$. With these definitions, the event

$S \geq k$ implies that the graph is completely covered. The Chernoff bound shows that $\Pr(S \leq (1-\varepsilon)pb) \leq e^{-bp\varepsilon^2/2}$, which for $\varepsilon = 1/2$ and our choices of b and p gives $e^{-\frac{t\alpha^k}{8k}}$. Finally, note that $(1-\varepsilon)pb = (1/2)\alpha^k t/k$ which is greater than k for our choice of t. Therefore, $\Pr(S < k) \leq \Pr(S < (1-\varepsilon)pb) \leq e^{-\frac{t\alpha^k}{8k}}$.

Finally, we need to show that $e^{-\frac{t\alpha^k}{8k}} < 1/(2t^c)$. It is enough to show that $-\frac{t\alpha^k}{8k} < -\ln(2t^c)$ or that $2t > \frac{16k}{\alpha^k} c \ln(2t)$. It can be easily verified that for $D \geq 3$ the equation $x > D \ln x$ is true for any $x \geq D^2$. In our case we take $D = \frac{16ck}{\alpha^k}$. (Note that $c > 1, k \geq 1$, and $\alpha < 1$; hence, indeed $D \geq 3$.) Therefore, the choice of t (with γ sufficiently large) guarantees the inequality. The lemma follows. □

Given that the walk does not use any α-light edge, and since every edge is either α-light or α-heavy, the probability that in step t the walk visits a state in which it is already visited is at least the probability that a walk of length $t-1$ completely covers the induced graph of α-heavy edges (since, by the definition of "completely covered," the only nodes that can be reached in step t, by an α-heavy edge, have already been visited).

LEMMA 9. *Consider a Markov chain $(Q_{i,j})$ such that each transition probability is either α-heavy or α-light. For any $c > 1$, the Markov chain is not (n, t^c)-live.*

Proof. In order to show that the Markov chain is not (n, t^c)-live, it is sufficient to show that there exists a t, such that the probability that in step t a new state is visited is less than $1/t^c$. The probability of reaching a new state at step t is

$$\begin{aligned}
&\Pr(\text{new state in step } t)\\
&= \Pr(\text{new state in step } t | \alpha\text{-light edge is used}) \cdot \Pr(\alpha\text{-light edge is used})\\
&+ \Pr(\text{new state in step } t | \text{no } \alpha\text{-light edge is used}) \cdot \Pr(\text{no } \alpha\text{-light edge is used})\\
&\leq \Pr(\alpha\text{-light edge is used}) + \Pr(\text{new state in step } t | \text{no } \alpha\text{-light edge is used}).
\end{aligned}$$

The first summand is less than $1/(2t^c)$, by Lemma 7. If the graph of the α-heavy edges is covered and no α-light edge was used, we cannot reach a new state. Therefore, the second summand is not more than the probability of not covering the graph in $t-1$ steps (given that no α-light edge is used). By Lemma 8, this probability is also less than $1/(2t^c)$. Altogether, we get that the probability of visiting a new state in step t is less than $1/t^c$. This implies that the Markov chain does not have the required liveness property. □

COROLLARY 10. *Let $c \geq 0$ be any constant. Let $(Q_{i,j})$ be any Markov chain on k states which is (n, t^c)-live. Then, $k = \Omega(\sqrt{\log\log n / \log\log\log n})$.*

Proof. Lemma 6 shows that if there is no "gap" of the form $(\alpha^{\lambda k}, \alpha]$, then the claimed lower bound holds. Lemma 9 shows that if there is such a "gap," then the Markov chain is not (n, t^c)-live. □

THEOREM 11. *Every mutual exclusion algorithm for n processes which guarantees t^c-fairness requires a shared variable of $\Omega(\log\log\log n)$ bits.*

Proof. The proof is similar to the proof of Theorem 4, but using Corollary 10 instead of Lemma 3. Consider the algorithm $\mathcal{A}$ and the corresponding Markov chain $Q(\mathcal{A}, d)$, and assume that the algorithm $\mathcal{A}$ guarantees t^c-fairness. If the Markov chain $Q(\mathcal{A}, d)$ is not $(n-1, 2t^c)$-live, then there exists a $1 \leq t \leq n-1$ which is $\frac{1}{2t^c}$-good. By Lemma 2, there exists an extension of the basic schedule such that the probability that P_t enters the critical section at either C_1 or C_2 is less than $\frac{1}{t^c}$, contradicting the t^c-fairness of the algorithm. Therefore, $Q(\mathcal{A}, d)$ must be $(n-1, 2t^c)$-live. By Corollary 10, this implies that k, the number of states in this

Markov chain, is $\Omega(\sqrt{\log\log n/\log\log\log n})$. By the construction of the Markov chain, the number of bits in the shared variable used by $\mathcal{A}$ is $\log(k-1) = \Omega(\log\log\log n)$, as claimed. □

To relax the requirement that the processes have the same program, we make the following observations. For every process P_i, we can associate with its program $\mathcal{A}_i$ a Markov chain $Q(\mathcal{A}_i, d)$, as before. All those Markov chains have the same number of states k. If $k = \Omega(\sqrt{\log\log n/\log\log\log n})$, we are done. That is, the number of bits of the shared variable is $\Omega(\log\log\log n)$. By Lemma 6, if k is "too small" (i.e., $k = o(\sqrt{\log\log n/\log\log\log n})$), then for every process there is some gap. That is, one of the k^2 intervals $(\beta_{\ell+1}, \beta_\ell]$ considered in the proof of Lemma 6 is empty. Moreover, for n/k^2 of the processes the gap is in the *same* interval. Denote this interval by $(\alpha^{\lambda k}, \alpha]$. Now, consider only these processes and the α-heavy edges in the corresponding graphs. The number of ways of choosing for each of the k^2 edges whether it is heavy or not is 2^{k^2}. Therefore, there are $n' \stackrel{\triangle}{=} n/(k^2 \cdot 2^{k^2})$ processes with the same gap, and the same α-heavy edges. If we take only these processes, the same proof can be repeated. Finally, note that due to the double logarithmic relation between k and n, the number of processes we remained with is

$$n' = n/(k^2 \cdot 2^{k^2}) > n^{\varepsilon}$$

for some constant $\varepsilon > 0$. Hence, we also get a lower bound of $\Omega(\log\log\log n') = \Omega(\log\log\log n)$ for the case when processes may use different programs. We conclude with the following theorem.

THEOREM 12. *Every mutual exclusion algorithm for n processes which guarantees t^c-fairness requires a shared variable of $\Omega(\log\log\log n)$ bits, even if each process runs a different program.*

5. Upper bounds. In this section, we present some upper bounds to complete the picture. In fact, we do not explicitly present protocols. Instead, we present appropriate *lotteries*, where a lottery is just a probability distribution that allows processes to draw numbers ("tickets") in $\{1, 2, \ldots, B\}$. The "winners" of the lottery are those processes drawing the maximal drawn number. We use as a *black box* the following theorem, implicit in [KR92], that reduces the existence of mutual exclusion algorithms with certain fairness properties to the existence of lotteries that guarantee a certain probability of having a *unique winner*.

THEOREM 13 (see [KR92]). *Let f be a function, and n and B be integers. Assume that there exists a lottery for at most n processes, on B values, with the property that for every number of processes $1 \le t \le n$ and every participating process P_i with probability at least $1/f(t)$, the maximal drawn number was drawn by the process P_i, and all other participating processes draw strictly smaller numbers. Then, there exists a randomized mutual exclusion algorithm for n processes that guarantees f-fairness and uses a shared variable of $2\log B + O(1)$ bits.*

By this theorem, in order to prove the existence of mutual exclusion algorithms, it is enough to prove the existence of the appropriate lotteries. For example, the lottery used in [Rab82, KR92] assigns a probability of 2^{-j} for each value $1 \le j < B$ ($B = 4 + \log n$), and probability 2^{-B+1} for the value B. It is shown that this lottery gives $f(t) = O(t)$ and therefore can be used to achieve mutual exclusion with linear fairness.

Note that all the upper bounds we give immediately give upper bounds on the number of states of (n, f)-live Markov chains for the appropriate f's. We start by showing an upper bound for a constant-size shared variable.

THEOREM 14. *There exists a randomized mutual exclusion algorithm that uses a constant-size shared variable and guarantees* $1/2^t$*-fairness.*

Proof. We show a lottery with $f(t) = 2^t$; by Theorem 13, this completes the proof. In the lottery, each participating process P_i chooses a value in $\{1, 2\}$ with uniform distribution; i.e., the probability that P_i chooses each of the two values is $1/2$. For every participating process P_i, we are interested in the event in which P_i chooses the maximum value and it is unique. Since there are only two possible values, this is simply the event in which P_i chooses the value 2 and all other participating processes choose 1. The probability that P_i chooses the value 2 and all other participating processes choose 1 is exactly $1/2^t$; therefore, $f(t) = 2^t$. ∎

The next theorem derives a bound in the case where the fairness guarantee needs to be polynomial in t (note that in the previous theorem, the fairness guarantee is exponential in t). We show the result by exhibiting a different lottery for this case. This lottery implies an upper bound of $O(\log\log\log n)$ bits for mutual exclusion with polynomial-fairness.

THEOREM 15. *For any constant* $c > 1$*, there exists a randomized mutual exclusion algorithm that uses an* $O(\log\log\log n)$*-size shared variable and guarantees* $\Omega(1/t^c)$*-fairness.*

Proof. Again, we show a lottery with $f(t) = O(t^c)$; by Theorem 13, this completes the proof. Consider the following lottery: the value j is chosen with probability $1/2^{c^j}$, for $(j = 1, 2, \ldots, c' \log\log n)$, and the value 0 is chosen otherwise. For every integer t (the number of participants), let $\ell \geq 0$ be an integer such that $2^{c^\ell} \leq t < 2^{c^{\ell+1}}$ (the constant c' is chosen so as to guarantee that such an ℓ exists for every $t \leq n$). We are interested in the event in which P_i chooses a value $\ell+1$ and all other $t-1$ participating processes choose values at most ℓ. This clearly lower-bounds the probability that P_i is the unique process that chooses the maximum value. The probability that P_i chooses the value $\ell + 1$ is $1/2^{c^{\ell+1}}$. For each P_j, $j \neq i$, the probability that P_j chooses a value greater than or equal to $\ell + 1$ is

$$\sum_{k=\ell+1}^{c' \log\log n} \frac{1}{2^{c^k}} \leq \sum_{k=\ell+1}^{c' \log\log n} \left(\frac{1}{2^c}\right)^k < \frac{c''}{2^{c^{\ell+1}}},$$

where c'' is a constant (e.g., $c'' = 2\lceil 1/\log_2 c \rceil$ suffices). Therefore, the probability that P_j chooses a value less than or equal to ℓ is at least $1 - \frac{c''}{2^{c^{\ell+1}}}$. Since we have $t-1$ different P_j's, the probability that P_i chooses $\ell + 1$ and all other $t - 1$ participating processes choose values of at most ℓ is at least

$$\frac{1}{2^{c^{\ell+1}}} \cdot \left(1 - \frac{c''}{2^{c^{\ell+1}}}\right)^{t-1} \geq \frac{1}{2^{c^{\ell+1}}} \cdot \left(1 - \frac{c''}{2^{c^{\ell+1}}}\right)^{2^{c^{\ell+1}}} > \frac{1}{2t^c} \cdot \left(\frac{1}{2e}\right)^{c''},$$

which completes the proof of the theorem. (The proof remains similar in the case when we wish to get a lottery with $f(t) = \alpha t^c$ for a *particular* constant α.) ∎

REFERENCES

[AS91] H. ATTIYA AND M. SNIR, *Better computing on the anonymous ring*, J. Algorithms, 12 (1991), pp. 204–238.

[Ben83] M. BEN-OR, *Another advantage of free choice: Complete asynchronous agreement protocols*, in Proc. 6th ACM Symp. on Principles of Distributed Computing, 1983, pp. 27–30.

[BFJ+82] J. E. Burns, M. J. Fischer, P. Jackson, N. A. Lynch, and G. L. Peterson, *Data requirements for implementation of n-process mutual exclusion using a single shared variable*, J. Assoc. Comput. Mach., 29 (1982), pp. 183–205.

[Bra85] G. Bracha, *An $O(\log n)$ expected rounds randomized byzantine generals protocol*, in Proc. 17th ACM Symp. on Theory of Computing, 1985, pp. 316–326.

[CIL87] B. Chor, A. Israeli, and M. Li, *On process coordination using asynchronous hardware*, in Proc. 6th ACM Symp. on Principles of Distributed Computing, 1987, pp. 86–97.

[Dij65] E. Dijkstra, *Solution of a problem in concurrent programming control*, Comm. ACM, 8 (1965), p. 569.

[FL82] M. Fischer and N. Lynch, *A lower bound for the time to assure interactive consistency*, Inform. Process. Lett., 14 (1982), pp. 183–186.

[FLP85] M. J. Fischer, N. A. Lynch, and M. S. Paterson, *Impossibility of distributed consensus with one faulty process*, J. Assoc. Comput. Mach., 32 (1985), pp. 374–382.

[FM88] P. Feldman and S. Micali, *Optimal algorithms for byzantine agreement*, in Proc. 20th ACM Symp. on Theory of Computing, 1985, pp. 148–161.

[GY89] R. L. Graham and A. C. Yao, *On the improbability of reaching byzantine agreements*, in Proc. 21st ACM Symp. on Theory of Computing, 1989, pp. 467–478.

[IR81] A. Itai and M. Rodeh, *The lord of the ring, or probabilistic methods for breaking symmetry in distributed networks*, in Proc. 22th IEEE Symp. on Foundations of Computer Science, 1981, pp. 150–158.

[KR92] E. Kushilevitz and M. O. Rabin, *Randomized mutual exclusion algorithms revisited*, in Proc. 11th ACM Symp. on Principles of Distributed Computing, 1992, pp. 275–283.

[KY84] A. Karlin and A. C. Yao, *Probabilistic Lower Bounds for Byzantine Agreement*, unpublished manuscript, 1984.

[LR81] D. Lehman and M. O. Rabin, *On the advantage of free choice: A symmetric and fully distributed solution to the dining philosophers problem*, in Proc. 8th ACM Symp. on Principles of Programming Languages, 1981, pp. 133–138.

[Rab82] M. O. Rabin, *n-process mutual exclusion with bounded waiting by $4\log_2 n$-valued shared variable*, J. Comput. System Sci., 25 (1982), pp. 66–75.

[Sai92] I. Saias, *Proving probabilistic correctness statements: The case of Rabin's algorithm for mutual exclusion*, in Proc. 11th ACM Symp. on Principles of Distributed Computing, 1992, pp. 263–272.

[Yao77] A. C. Yao, *Probabilistic computations: Toward a unified measure of complexity*, in Proc. 18th IEEE Symp. on Foundations of Computer Science, 1977, pp. 222–227.

SIAM J. COMPUT.
Vol. 27, No. 6, pp. 1564–1591, December 1998
© 1998 Society for Industrial and Applied Mathematics
004

ADAPTIVE HEURISTICS FOR BINARY SEARCH TREES AND CONSTANT LINKAGE COST*

TONY W. LAI† AND DERICK WOOD‡

Abstract. We present lower and upper bounds on adaptive heuristics for maintaining binary search trees using a constant number of link or pointer changes for each operation (constant linkage cost (CLC)). We show that no adaptive heuristic with an amortized linkage cost of $o(\log n)$ can be competitive. In particular, we show that any heuristic that performs $f(n) = o(\log n)$ promotions (rotations) amortized over each access has a competitive ratio of at least $\Omega(\log n/f(n))$ against an oblivious adversary, and any heuristic that performs $f(n) = o(\log n)$ pointer changes amortized over each access has a competitive ratio of at least $\Omega(\frac{\log n}{f(n)\log(\log n/f(n))})$ against an adaptive online adversary.

In our investigation of upper bounds we present four adaptive heuristics:

- a randomized, worst-case-CLC heuristic randomized two-promotion (R2P) whose expected search time is within a constant factor of the search time using an optimal tree; that is, it is statically competitive against an oblivious adversary;
- a randomized, expected-CLC heuristic (locally optimized randomized partial splay (LORPS)) that has $O(\log n)$ expected-amortized update time and is statically competitive against an oblivious adversary;
- a deterministic, amortized-CLC heuristic (locally optimized partial splay (LOPS)) that has $O(\log n)$ amortized update time and is statically competitive against an adaptive adversary;
- a practical, randomized heuristic (randomized partial splay (RPS)) that is not CLC but has performance bounds comparable with those of the splay heuristic of Sleator and Tarjan; it is statically competitive against an adaptive adversary.

The randomized heuristics use only constant extra space, whereas the deterministic heuristic uses $O(n)$ extra space.

Key words. adaptivity, self-adjustment, self-organization, binary search trees, constant linkage cost, expected amortization, competitive ratio

AMS subject classifications. 68P05, 68P10, 68O25, 68R99

PII. S0097539793250329

1. Introduction.

One of the most fundamental problems in computer science is the *dictionary problem:* we have a totally ordered universe U, and we want to maintain efficiently a set $S \subseteq U$ that supports the operations of *insert, delete,* and *member.* The *binary search tree* is a well-known and well-studied data structure that can support these operations in $O(\log n)$ time in the expected case. Many kinds of balanced trees have been proposed that can support all dictionary operations in $O(\log n)$ worst-case time [1, 11, 16]; however, these structures cannot adapt to nonuniform or skewed access patterns. We consider the problem of maintaining an *adaptive binary search tree* (also known as a self-organizing search tree [2] and a self-adjusting search tree [21]).

*Received by the editors June 15, 1993; accepted for publication (in revised form) August 19, 1996; published electronically June 3, 1998. A preliminary version of this paper appeared in *Proc. 2nd Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, 1991, pp. 72–77.
http://www.siam.org/journals/sicomp/27-3/25032.html

†IBM Canada Ltd., 1150 Eglinton Avenue East, North York, Ontario M3C 1H7, Canada (twhlai@vnet.ibm.com). The work of this author was partially supported by a NSERC postgraduate scholarship.

‡Department of Computer Science, Hong Kong University of Science & Technology, Clear Water Bay, Kowloon, Hong Kong (dwood@cs.ust.hk). The work of this author was supported by grants from the Natural Sciences and Engineering Research Council of Canada and the Information Technology Research Centre of Ontario.

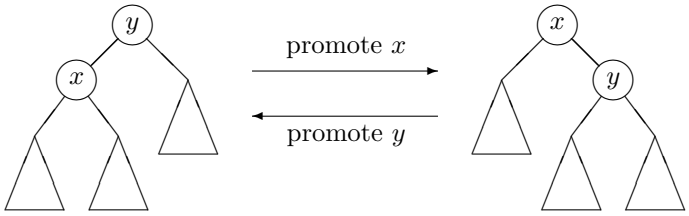
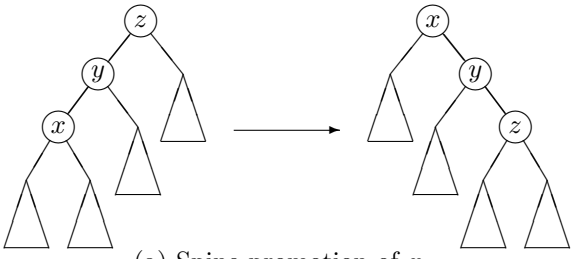
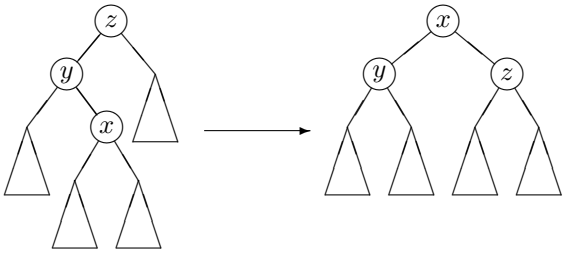


FIG. 1. *The promotion operation.*



(a) Spine promotion of x



(b) Zig-zag promotion of x

FIG. 2. *The two-promotion operations.*

Compared with balanced trees, adaptive trees have the disadvantage of requiring more restructuring, especially during accesses (searches). This additional work often makes adaptive trees less practical than balanced trees; for instance, the performance of splay trees [21] is worse than that of AVL trees unless the pattern of accesses is very highly skewed [4]. High overhead, though, is not inherent to adaptation in general. For instance, the move-to-front heuristic for sequential lists changes only a constant number of pointers for each operation, yet is very efficient in practice [7]. To restructure a tree, we use the promotion operation (see Figure 1), also known as a rotation, and the two-promotion operation (see Figure 2), also known as a splaying step.

We consider the problem of minimizing the restructuring that is required for each operation. Ideally, we would like adaptivity to have the following four properties:

1. It should change at most a constant number of pointers for each operation in the worst case; that is, it should have *CLC.*
2. It should support both accesses and updates.
3. Its running time should be within a constant factor of the time of any heuristic for sufficiently long sequences of operations; that is, it should be *competitive.*

4. It should use only constant extra space.

We are interested in the CLC restriction for at least three reasons. First, our main motivation is that CLC is crucial when binary search trees have adjunct secondary structures such as in priority–search trees [14], segment–range trees [20], and segment–segment trees [20]. (A weaker motivation is the storage of binary search trees on secondary memory when the CLC constraint implies that only a constant number of blocks have to be rewritten after an access.) In each case, a promotion in a primary binary search tree of size n typically causes an $\Omega(\log n)$-time update in its adjunct structure; therefore, $\Omega(k)$ promotions take $\Omega(k \log n)$ time. Since we want updates to take only $O(\log n)$ time in most circumstances, CLC updates are appealing. It is well known that insertions into an AVL tree are CLC, whereas deletions may cause $\Theta(\log n)$ promotions in an AVL tree of size n. This behavior is the reason for using red–black trees to implement priority–search trees since red–black trees have both CLC insertions and CLC deletions.

Second, the CLC constraint is, we believe, fundamental and should be investigated. Ottmann and Wood [17], for example, present a general framework for obtaining binary search trees that are CLC in the worst case. Third, it may provide insight into the dynamic optimality conjecture for splay trees (see the work of Cole [10]).

Various schemes for maintaining adaptive search trees have been proposed such as move-to-root [2], simple exchange [2], monotonicity [8], dynamic binary search [15], splaying [21], biasing [6], conditional rotation [9], weighted randomization [3], and deepsplaying [19]. Of these schemes, only simple exchange, conditional rotation, and weighted randomization are applicable to binary search trees and also are CLC. Simple exchange and conditional rotation are worst-case CLC but have an amortized access time of $\Theta(n)$. In contrast, weighted randomization is expected CLC and has an expected access time comparable with that of an optimal static tree for fixed access sequences. Simple exchange does not use extra space, whereas conditional rotation and weighted randomization use $O(n \log m)$ bits of extra space, where m is the number of performed accesses.

While some upper bounds are known for adaptive, constant-linkage-cost heuristics [2, 3, 9], few lower bounds are known. In section 3, we show that no worst-case-CLC heuristic can be competitive. Indeed, an online adaptive adversary can force any heuristic with an amortized linkage cost of $f(n) = o(\log n)$ to use a multiplicative factor of $\Omega(\frac{\log n}{f(n) \log(\log n / f(n))})$ more time than the splay heuristic to access an arbitrarily long sequence. In light of this lower bound, we replace property 3 with property 3′.

3′. The heuristic's access time should be within a constant factor of the time taken by any static tree for any sufficiently long access sequence; that is, the heuristic should be *statically competitive.*

Although this revised property may be unnecessarily weak, it can be considered both a minimum requirement for effective adaptation and a first step in the investigation of CLC adaptive heuristics. We leave as an open problem the investigation of lower bounds on competitiveness of arbitrary heuristics against an oblivious adversary.

Although we have not been able to improve on the results for splay trees under amortization, we can improve them with respect to the expected time taken by a sequence of operations. (Just as we can bound the greatest (or worst) and expected times taken by a single operation, we can also bound the greatest (or worst) and expected times taken by sequences of operations.) We present three statically competitive CLC heuristics and a statically competitive heuristic that is not CLC yet has performance bounds comparable with Sleator and Tarjan's splay trees. There are

TABLE 1
A comparison of the R2P, LORPS, LOPS, and RPS heuristics. LOPS is a deterministic heuristic, whereas R2P, LORPS, and RPS are randomized heuristics.

Heuristic	Linkage cost	Statically competitive	Update time	Extra space
R2P	$O(1)$ worst case	yes/obliv	$\Theta(n)$	$O(1)$
LORPS	$O(1)$ expected	yes/obliv	$O(\log n)$ expect	$O(1)$
LOPS	$O(1)$ amortized	yes/adap	$O(\log n)$ amort	$O(n)$
RPS	$O(\log n)$ amortized	yes/adap	$O(\log n)$ amort	$O(1)$

two kinds of results: we focus on accesses and consider static competitiveness, and we consider all three dictionary operations and compare the performance of the heuristics with the performance of red–black and splay trees.

The first (the R2P heuristic) is a randomized, worst-case-CLC heuristic that supports only accesses. The second (the LORPS heuristic) is a randomized, expected-CLC heuristic that supports updates in $O(\log n)$ expected time. The third (the LOPS heuristic) is a deterministic, amortized-CLC heuristic that also supports updates in $O(\log n)$ amortized time. The randomized heuristics use only constant extra space, or $O(\log n)$ bits of extra space in a bit-cost model, whereas the deterministic heuristic uses $O(n)$ extra space, or $O(n \log n)$ bits of extra space in a bit-cost model. The last (the RPS heuristic) is a new, practical, randomized heuristic that is not CLC but has a simple, efficient implementation and can be preferable to the splay heuristic when the pattern of accesses is stable. RPS supports updates in $O(\log n)$ amortized time, and it has $O(\log n)$-amortized linkage cost. A comparison of the heuristics is shown in Table 1.

2. Definitions. We define the *depth* or *level* of a node x of a tree to be the number of edges in the root-to-x path; the depth of the root is 0. We define (x, y) to be a *left edge* in a tree T if y is a left child of x in T. We define a *right edge* analogously.

In our analyses, we assign positive numbers called *weights* to each node in a tree. We define the *size* $s(x)$ of a node x to be the sum of the weights of all nodes in the subtree rooted at x. The *rank* $r(x)$ of x is $\log s(x)$. The *total rank* $R(T)$ of a tree T is the sum of all ranks in the tree. The *weighted path length* of a tree T is 0 if T is empty or the sum of all weights in T plus the weighted path lengths of T's left and right subtrees otherwise.

When analyzing an adaptive algorithm, we assume that it services requests generated by an adversary. An *oblivious adversary* generates a fixed sequence of requests in advance, without knowledge of the state of the adaptive algorithm. An *adaptive adversary* generates a request after seeing the effect of all previous requests on the state of the algorithm. Notice that when an adaptive algorithm is deterministic, adaptive and oblivious adversaries have the same power. We use this observation when analyzing the LOPS heuristic.

We may also compare the performance of an adaptive algorithm with the performance of an adversary. An *online adversary* services each request immediately after generating it, whereas an *offline adversary* can service requests after generating the entire request sequence. We thus have three distinct types of adversaries: oblivious, adaptive online, and adaptive offline. For more details, consult the work of Ben-David et al. [5].

Let $C_H(r_1, \ldots, r_m)$ be the cost incurred by an adaptive heuristic H when servicing the request sequence $r_1, \ldots, r_m$. Let $C_A(r_1, \ldots, r_m)$ be the cost incurred by an adversary A when servicing $r_1, \ldots, r_m$. We say that H is *c-competitive* against A if there is a real number e, independent of m, such that

$$E[C_H(r_1, \ldots, r_m) - c \cdot C_A(r_1, \ldots, r_m)] \leq e$$

for any finite sequence $r_1, \ldots, r_m$ generated by A. The *competitive ratio* of H is infimum over all c such that H is c-competitive.

Note that, in the definition of competitiveness, we use the same cost model for the heuristic H and the adversary A. We handicap the adversary by giving it a less advantageous cost model. Let $C_S(r_1, \ldots, r_m)$ be the cost incurred by a static structure (chosen by the adversary before it services the requests) for the request sequence $r_1, \ldots, r_m$. We say that an adaptive heuristic H is *statically c-competitive* against an adversary A, if there is a real number e, independent of m, such that

$$E[C_H(r_1, \ldots, r_m) - c \cdot C_S(r_1, \ldots, r_m)] \leq e,$$

for any finite sequence $r_1, \ldots, r_m$ of accesses generated by A, where the cost models for A and H are not necessarily the same. If H is statically c-competitive for some constant c, then we say that H is *statically competitive.* Note that we have outlawed updates within the framework of static competitiveness.

Let Φ be a potential function that maps each configuration of some data structure to a real number. We define the *expected-amortized time* a of an operation to be $E[t + \Phi(D') - \Phi(D)]$, where t is the actual time of the operation, and D and D' are the configurations of the data structure before and after the operation, respectively. Since E is a linear operator, $E[t] = a + E[\Phi(D)] - E[\Phi(D')]$; hence, we can show that the expected time of m operations is the sum of the expected-amortized times of the operations and the expected drop in potential. In other words,

$$E\left[\sum_{i=1}^{m} t_i\right] = E\left[\Phi_0 - \Phi_m + \sum_{i=1}^{m} a_i\right],$$

where t_i is the actual time of the ith operation, a_i is the expected-amortized time of the ith operation, Φ_0 is the initial potential, and Φ_i is the potential after the ith operation. By bounding the maximum drop in potential, we can use the expected-amortized time to obtain an upper bound on the expected time of a sequence of operations. This technique is analogous to the potential method of conventional amortized analysis. Lai [12] discusses expected-amortized time more thoroughly as do Ben-David et al. [5] and Raghavan and Snir [18].

3. Lower bounds. We use a pointer machine as the model of computation in the proofs of the lower bounds. We assume without loss of generality that a heuristic performs restructuring only after it searches for an accessed element.

We present two lower bounds. The first bound applies to a restricted model in which only promotions are used to restructure a tree, whereas the second bound applies to arbitrary pointer changes. The first model applies to randomized search trees [3] and establishes that although the heuristic is expected CLC, it cannot be competitive.

3.1. The restricted model. We prove that, for any function $f(n) = o(\log n)$, if a heuristic H performs at most $f(n)$ promotions amortized over accesses, then H has a competitive ratio of $\Omega(\log n/f(n))$ against an oblivious adversary. In the following, we say that a heuristic is *promotion based* if it uses only promotions to alter the shape of a tree.

To prove our lower bound on H's competitive ratio, the adversary maintains its tree using the splay heuristic [21]. The splay heuristic has three important properties: it has an amortized access time of $O(\log n)$, it moves accessed elements to the root, and it does not change the tree if the root element is accessed. Thus, the amortized time of accessing some element r times consecutively, using the splay heuristic, is $O(\log n + r)$.

The heuristic H, on the other hand, cannot quickly move elements to the root. In general, after $\log n/(2f(n))$ accesses, H cannot perform more than $\log n/2$ promotions. Thus, if an adversary knows an element x of depth at least $\log n$ in H's tree, and accesses it $r \leq \log n/(2f(n))$ times, then x will still have depth $\Omega(\log n)$ afterward, which implies that H requires $\Omega(r \log n)$ time. Using this line of reasoning, we can easily establish a lower bound of $\Omega(\log n/f(n))$ on the competitive ratio of H against an adaptive, online adversary that uses the splay heuristic.

To show that an oblivious adversary can force poor performance from H, we show that an oblivious adversary can consistently access nodes of great depth in a tree that is maintained by any promotion-based heuristic. Let $\mathrm{rev}_k(i)$ be the number resulting from reversing the bits of a k-bit number i. Wilber [22] proved that if $n = 2^k$, then there exists a fixed sequence of length n, namely $\langle \mathrm{rev}_k(0), \ldots, \mathrm{rev}_k(n-1)\rangle$, that requires $\Omega(n \log n)$ access time in a search tree of size n for the n keys $0 \ldots n-1$, using any promotion-based heuristic.

LEMMA 3.1 (Wilber). *Let T be a tree of size $n = 2^k$ that contains nodes labeled $0, \ldots, n-1$. Assume that we maintain T using an adaptive promotion-based heuristic, and that we are allowed to access only the root element. The time to access the sequence $S_1 = \langle \mathrm{rev}_k(0), \ldots, \mathrm{rev}_k(n-1)\rangle$ is at least $n \log n + 1$; that is, the number of promotions required to access S_1 is at least $n \log n - n + 1$.*

Wilber's lower bound can be easily extended for the case where n is not a power of 2; it is straightforward to extend Wilber's lower bound to show that, for any $0 \leq k \leq \lfloor \log n \rfloor$, the number of promotions to access the sequence $\langle \mathrm{rev}_k(0), \ldots, \mathrm{rev}_k(2^k - 1)\rangle$ is at least $k \cdot 2^k - 2^k + 1$.

Wilber observed that limiting accesses to only the root is not a serious limitation, since we can access a node of depth d by promoting it to the root and returning it to its original position, at the expense of performing $2d$ extra promotions. That is, if we have a general heuristic that can access S_1 in A time using P promotions, then we can transform it into a heuristic that performs only root accesses and uses at most $2A - n + P$ promotions to access S_1. Thus, Wilber's lower bound implies that there is a tradeoff between the access time and the number of promotions performed for general promotion-based heuristics, as we now claim.

COROLLARY 3.2. *Let T be a tree of size n that contains nodes labeled $0, \ldots, n-1$, let $k = \lfloor \log n \rfloor$, and let $N = 2^k$. Assume that we maintain T using an adaptive promotion-based heuristic. If we access the sequence $S_1 = \langle \mathrm{rev}_k(0), \ldots, \mathrm{rev}_k(N-1)\rangle$, then the access time A and number P of promotions performed satisfies $2A + P \geq N \log N + 1$.*

We prove our main theorem on the competitiveness of H as follows. Let S_r be the sequence obtained from the sequence $S_1 = \langle \mathrm{rev}_k(0), \ldots, \mathrm{rev}_k(2^k - 1)\rangle$ by repeating

each element r times. We note that the splay heuristic requires $\Theta(n \log n)$ time to access S_r, whereas we show that heuristic H requires $\Omega(rn \log n)$ time, provided that $r = \Theta(\log n / f(n))$. Finally, we note that if a heuristic H performs $f(n)$ promotions amortized over accesses, for some function f, then there exists a function g such that H performs at most $g(n) + m \cdot f(n)$ promotions in its first m accesses for all m.

THEOREM 3.3. *For any functions f and g such that $f(n) = o(\log n)$, if a promotion-based heuristic H performs at most $g(n) + m \cdot f(n)$ promotions in its first m accesses, for all m, then H has a competitive ratio of $\Omega(\log n / f(n))$ against an oblivious adversary.*

Proof. Without loss of generality, assume that the elements are labeled $0, \ldots, n-1$. Let $N = 2^{\lfloor \log n \rfloor}$ and, for $0 \le i \le N-1$, let $x_i = \mathrm{rev}_{\lfloor \log n \rfloor}(i)$. Let $r = \lfloor \log N / (8f(n)) \rfloor$ and let $S_r = \langle x_0 \rangle^r \langle x_1 \rangle^r \ldots \langle x_{N-1} \rangle^r$. We assume that n is large enough such that $r \ge 1$. Since r has been chosen to be $O(\log n)$, the splay heuristic takes time $O(n \log n)$ to access S_r. We claim that, for all $q \ge \max(2g(n), 1)$, the time to access $(S_r)^q$ using heuristic H is $\Omega(qn \log^2 n / f(n))$. Since the splay heuristic can access $(S_r)^q$ in $O(qn \log n)$ time, the claim implies that H has a competitive ratio of $\Omega(\log n / f(n))$. To prove the claim, it is sufficient to show the following two facts. First, there are at least $\lceil q/2 \rceil$ accesses of S_r in which we perform at most $N \log N/4 + 1$ promotions. Second, if we perform at most $N \log N/4 + 1$ promotions when accessing S_r, then the access time is $\Omega(N \log^2 N / f(n))$.

Because the sequence $(S_r)^q$ is of length qrN, the heuristic H performs at most $g(n) + qrN \cdot f(n)$ promotions when accessing $(S_r)^q$. Observe that there must be at least $\lceil q/2 \rceil$ accesses of S_r in which H performs no more than $2\left[g(n) + qrN \cdot f(n)\right]/q = 2g(n)/q + 2rN \cdot f(n)$ promotions. Since $q \ge 2g(n)$, we have $2rN \cdot f(n) + 2g(n)/q \le 2rN \cdot f(n) + 1 \le N \log N/4 + 1$.

It remains to show that if we access S_r using at most $N \log N/4 + 1$ promotions, then the access time is $\Omega(n \log^2 n / f(n))$. For $0 \le i \le N-1$, let t_i be the time required for the first access of x_i in S_r and, for $0 \le i \le N-1$ and $1 \le j \le r$, let p_{ij} be the number of promotions performed after accessing the jth occurrence of x_i. Consider a heuristic H' that performs the same promotions when accessing x_i as the heuristic H does when accessing $\langle x_i \rangle^r$ for all i. Observe that the access time of H' over S_1 is exactly $\sum_{i=0}^{N-1} t_i$. Since H' performs at most $N \log N/4 + 1$ promotions when accessing $S_1 = x_0 x_1 \ldots x_{N-1}$, Corollary 3.2 implies that $\sum_{i=0}^{N-1} t_i \ge 3N \log N/8$. Finally, observe that the time to access the jth occurrence of x_i is at least $t_i - \sum_{k=1}^{j-1} p_{ik}$, since we can move x_i upward at most $\sum_{k=1}^{j-1} p_{ik}$ levels after x_i was first accessed. Therefore, the access time A is at least

$$\begin{aligned}\sum_{i=0}^{N-1} \sum_{j=1}^{r} \left[t_i - \sum_{k=1}^{j-1} p_{ik} \right] &= \sum_{i=0}^{N-1} \sum_{j=1}^{r} t_i - \sum_{i=0}^{N-1} \sum_{j=1}^{r} \sum_{k=1}^{j-1} p_{ik} \\ &= r \cdot \sum_{i=0}^{N-1} t_i - \sum_{i=0}^{N-1} \sum_{j=1}^{r} \sum_{k=1}^{j-1} p_{ik}.\end{aligned}$$

Since $\sum_{i=0}^{N-1} \sum_{j=1}^{r} p_{ij} \le N \log N/4 + 1$, we have

$$\sum_{i=0}^{N-1} \sum_{j=1}^{r} \sum_{k=1}^{j-1} p_{ik} \le \frac{rN \log N}{4} + r;$$

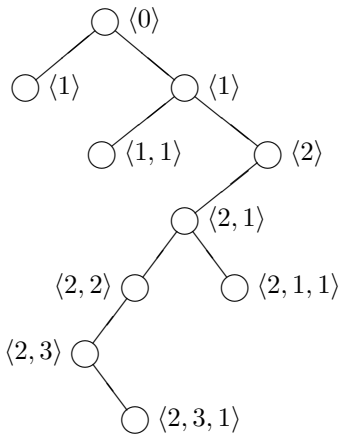


FIG. 3. *Turn sequences in a tree.*

thus,

$$A \geq r \cdot \frac{3N \log N}{8} - r \cdot \frac{N \log N}{4} - r$$
$$= \Omega\left(\frac{N \log^2 N}{f(n)}\right). \qquad \square$$

3.2. The general model. We now consider the general case in which we may detach and attach arbitrary edges to modify the shape of the tree. We prove that, for any functions $g(n)$ and $f(n) = o(\log n)$, if a heuristic H attaches and detaches at most $g(n) + m \cdot f(n)$ pointers in its first m accesses, for all m, then H has a competitive ratio of at least $\Omega(\frac{\log n}{f(n) \log(\log n / f(n))})$ against an adaptive, online adversary.

Because we allow arbitrary pointer changes, the lower bound of Wilber does not help us. Instead, we analyze what we call the *turn sequences* of nodes. We say that a node x has turn sequence $\langle a_1, \ldots, a_k \rangle$ if the root-to-x path consists of either a_1 left edges, a_2 right edges, a_3 left edges, and so forth, or a_1 right edges, a_2 left edges, a_3 right edges, and so forth; see Figure 3. We also call each path of left edges and each path of right edges a *segment*. More formally, we define a turn sequence as follows.

DEFINITION 3.4. *Let x be a node on level l in a tree T. Let $U = \langle a_1, a_2, \ldots, a_t \rangle$ be a sequence of nonnegative integers. For $0 \leq i \leq l$, let x_i be the ancestor of x on level i. Let $h(i) = j$ if $\sum_{k=1}^{j-1} a_k \leq i < \sum_{k=1}^{j} a_k$. We say that x has* turn sequence U *if one of the following three statements holds.*

1. *x is the root of T, $a_1 = 0$, and $t = 1$.*
2. *$l = \sum_{i=1}^{t} a_i$; for all i, $1 \leq i \leq t$, $a_i \geq 1$; and, for all i, $0 \leq i < l$ (x_i, x_{i+1}) is a left edge if and only if $h(i)$ is even.*
3. *$l = \sum_{i=1}^{t} a_i$; for all i, $1 \leq i \leq t$, $a_i \geq 1$; and, for all i, $0 \leq i < l$ (x_i, x_{i+1}) is a left edge if and only if $h(i)$ is odd.*

If x has turn sequence U, then we say that the root-to-x path has $t-1$ turns*, or x has $t-1$* turns*. We also define the jth* segment *to be the set of edges $\{(x_i, x_{i+1}) : h(i) = j\}$.*

To prove our lower bound, we first characterize how quickly we can move a node closer to the root; we show that one pointer change can shorten a path to a node by

at most one segment. We then use a combinatorial argument to prove that, for any tree T, either the height of T is large or there is a node x in T that has depth $\Omega(\log n)$ even if we remove many segments in the root-to-x path. Thus, for any heuristic H of linkage cost $o(\log n)$, an adversary can always access an element x some number of times r such that the total access time of $\langle x \rangle^r$ using heuristic H is $\omega(\log n)$, while the amortized access time of $\langle x \rangle^r$ using the splay heuristic is $O(\log n)$.

LEMMA 3.5. *Suppose that a node x on level l in a tree T has turn sequence $\langle a_1, \ldots, a_k \rangle$. For all j, $1 \le j \le k$, let b_j be the jth smallest element of the multiset $\{a_i : 1 \le i \le k\}$. If p pointers in T are detached and p new pointers are attached to obtain a new tree T', then the depth of x in T' is at least $\sum_{i=1}^{k-p} b_i$.*

Proof. By definition of a segment, the root-to-x path in T has k segments $S_1, \ldots, S_k$. Since p pointers are detached to obtain T', at least $k-p$ segments are unaffected and occur in both T and T'. To prove that the depth of x in T' is at least $\sum_{i=1}^{k-p} b_i$, it is sufficient to show that, for any j, if the edges in a segment S_j are in T', then S_j is in the root-to-x path in T', or the lowest node z in S_j is an ancestor of x. Consider the bottom edge (y, z) of S_j. If $z \ne x$, then by definition of a segment, x falls between y and z in an inorder traversal of T and, thus, of T', which implies that z must be an ancestor of x. □

Let n_{ij} be the maximum possible number of nodes on level i with j turns in a tree and N_{ht} be the maximum possible number of nodes of a tree of height h such that each node has fewer than t turns. By definition, a tree of height h has nodes of depth at most $h-1$, so $N_{ht} \le \sum_{i=0}^{h-1} \sum_{j=0}^{t-1} n_{ij}$. We now relate the height of a tree T with the size of T and the maximum number of turns in T.

LEMMA 3.6. *If $t \le h/3$, then $N_{ht} < 1 + 4\binom{h-1}{t}$.*

Proof. Observe that $n_{00} = 1$ and $n_{10} = 2$. Also note that, for $i > 1$, a node on level i with t turns must have a parent on level $i-1$ with either t or $t-1$ turns if $t > 0$. Hence, for $i > 1$ and $0 \le j \le i-1$, we have $n_{ij} \le n_{i-1,j-1} + n_{i-1,j}$, where $n_{k,-1} = n_{k,k} = 0$ for all k. By induction, we can show that $n_{ij} \le 2\binom{i-1}{j}$ for $i \ge 1$ and $0 \le j \le i-1$.

Therefore, the maximum number N_{ht} of nodes in a tree of height h in which each node has fewer than t turns is

$$\begin{aligned}
\sum_{i=0}^{h-1} \sum_{j=0}^{t-1} n_{ij} &= 1 + \sum_{i=1}^{h-1} \sum_{j=0}^{t-1} n_{ij} \\
&\le 1 + 2 \sum_{i=1}^{h-1} \sum_{j=0}^{t-1} \binom{i-1}{j} \\
&= 1 + 2 \sum_{j=0}^{t-1} \sum_{i=0}^{h-2} \binom{i}{j} \\
&= 1 + 2 \sum_{j=0}^{t-1} \binom{h-1}{j+1}.
\end{aligned}$$

Since $t \le h/3$, we have, for all $k \le t$,

$$\binom{h-1}{k-1} \le \frac{1}{2} \binom{h-1}{k}.$$

Thus,

$$N_{ht} \le 1 + 2\sum_{j=1}^{t} 2^{j-t}\binom{h-1}{t} < 1 + 4\binom{h-1}{t}. \qquad \square$$

COROLLARY 3.7. *If $t \le h/3$, then $N_{h+1,t} < 1 + 4(eh/t)^t$.*

Proof. The proof is immediate from Lemma 3.6 and the fact that $(h-1)\cdots(h-t) < h^t$ and $t! \ge t^t e^{-t}$. $\square$

Corollary 3.7 implies that if the maximum number of turns in a tree T of size n is small, then the height of T must be large compared with $\log n$. We use this fact to prove that, for any tree T and any function $f(n) = o(\log n)$, either T is of height $\Omega(\log^2 n/f(n))$ or there is a node x in T such that the depth of x is $\Omega(\log n)$ even if we remove $O(\log n/\log(\log n/f(n)))$ segments in the root-to-x path.

LEMMA 3.8. *For any function $f(n) = o(\log n)$ and any tree T of sufficiently large size n, one of the following two statements holds.*

1. *T is of height at least $\log^2 n/f(n)$.*
2. *There is a node with turn sequence $\langle a_1, \ldots, a_k\rangle$ such that $k \ge p$ and $\sum_{i=1}^{k-p} b_i \ge \log n/4$, where b_i is the ith smallest element of the multiset $\{a_1, \ldots, a_k\}$, and $p = \lfloor \frac{\log n}{8[4+\log e+\log(\log n/f(n))]} \rfloor$.*

Proof. We prove the lemma by contradiction. Because at most two nodes have the same turn sequence, the n nodes of T must have at least $n/2$ distinct turn sequences. Hence, it is sufficient to show that there are at most $o(n)$ distinct turn sequences $\langle a_1, \ldots, a_k\rangle$ such that $a_1 + \cdots + a_k < \log^2 n/f(n)$ and $k < p$ or $\sum_{i=1}^{k-p} b_i < \log n/4$.

Let $H = \lceil \log^2 n/f(n)\rceil - 1$. Corollary 3.7 implies that the number of distinct turn sequences $\langle a_1, \ldots, a_k\rangle$ such that $a_1 + \cdots + a_k \le H$ and $k < p$ is at most $N_{H+1,p} < 1 + 4(\frac{eH}{p})^p < 1 + 4(\frac{e\log^2 n}{f(n)p})^p$. We claim that $N_{H+1,p} < 1 + 4n^{1/4}$. It is sufficient to show that $p\log(\frac{e\log^2 n}{f(n)p}) < \log n/4$. Note that, by the definition of p, we have $p \ge \frac{\log n}{16[4+\log e+\log(\log n/f(n))]}$ for sufficiently large n. Now,

$$\begin{aligned} p\log\left(\frac{e\log^2 n}{f(n)p}\right) &\le p\log\left(\frac{e\log^2 n}{f(n)}\cdot\frac{16[4+\log e+\log(\frac{\log n}{f(n)})]}{\log n}\right) \\ &= p\log\left(16e\cdot\frac{\log n}{f(n)}\left[4+\log e+\log\left(\frac{\log n}{f(n)}\right)\right]\right) \\ &= p\left[4+\log e+\log\left(\frac{\log n}{f(n)}\right)+\log\left(4+\log e+\log\left(\frac{\log n}{f(n)}\right)\right)\right] \\ &< \frac{\log n}{8[4+\log e+\log(\frac{\log n}{f(n)})]}\cdot 2\cdot\left[4+\log e+\log\left(\frac{\log n}{f(n)}\right)\right] \\ &= \frac{\log n}{4}. \end{aligned}$$

Observe that any turn sequence $\langle a_1, \ldots, a_k\rangle$ such that $a_1 + \cdots + a_k \le H$ and $\sum_{i=1}^{k-p} b_i < \log n/4$ can be constructed by interleaving two sequences $\langle c_1, \ldots, c_p\rangle$ and $\langle d_1, \ldots, d_{k-p}\rangle$ sequences such that $\sum_{i=1}^{k-p} d_i \le M$, where $M = \lceil \log n/4\rceil - 1$. The number t of such turn sequences $\langle a_1, \ldots, a_k\rangle$ is at most the product of the number of turn

sequences $\langle c_1, \ldots, c_p\rangle$, the number of turn sequences $\langle d_1, \ldots, d_{k-p}\rangle$, and the number of interleavings of $\langle c_1, \ldots, c_p\rangle$ and $\langle d_1, \ldots, d_{k-p}\rangle$, summed over all k. Clearly, the number of turn sequences $\langle c_1, \ldots, c_p\rangle$ is at most $N_{H+1,p} \leq 1+4n^{1/4}$ since $\sum_{i=1}^{p} c_i < H+1$. Also, there are at most $\binom{M}{k-p}$ sequences $\langle d_1, \ldots, d_{k-p}\rangle$ such that $d_1 + \cdots + d_{k-p} \leq M$ since $d_1, \ldots, d_{k-p} \geq 1$. The number of interleavings of $\langle c_1, \ldots, c_p\rangle$ and $\langle d_1, \ldots, d_{k-p}\rangle$ is $\binom{k}{p}$. Note that $p \leq k$ and $k - p \leq M$, which imply that $p \leq k \leq p + M$. Therefore, the number t of possible turn sequences $\langle a_1, \ldots, a_k\rangle$ such that $a_1 + \cdots + a_k \leq H$ and $\sum_{i=1}^{k-p} b_i \leq M$ is at most

$$\begin{aligned}\sum_{k=p}^{p+M}\left[(1+4n^{1/4})\binom{M}{k-p}\binom{k}{p}\right] &= (1+4n^{1/4})\sum_{j=0}^{M}\left[\binom{M}{j}\binom{j+p}{p}\right] \\ &< (1+4n^{1/4})\binom{M+p}{p}\sum_{j=0}^{M}\binom{M}{j} \\ &= (1+4n^{1/4})\binom{M+p}{p}2^M.\end{aligned}$$

Since $M + p < H$ for large n, we know that $\binom{M+p}{p} < \binom{H}{p} < (\frac{eH}{p})^p < n^{1/4}$. Also, $2^M < 2^{\log n/4} = n^{1/4}$. Thus, $t = O(n^{1/4} \cdot n^{1/4} \cdot n^{1/4}) = O(n^{3/4})$. Therefore, there are at most $O(n^{3/4}) = o(n)$ distinct turn sequences $\langle a_1, \ldots, a_k\rangle$ such that $a_1 + \cdots + a_k < \log^2 n/f(n)$, and $k < p$ or $\sum_{i=1}^{k-p} b_i < \log n/4$, which completes the proof. □

The proof of the lower bound for arbitrary pointer changes is now straightforward.

THEOREM 3.9. *For any functions f and g such that $f(n) = o(\log n)$, if a heuristic H performs at most $g(n) + m \cdot f(n)$ pointer changes during its first m accesses, for all m, then H has a competitive ratio of $\Omega(\frac{\log n}{f(n)\log(\log n/f(n))})$ against adaptive, online adversaries.*

Proof. Let $p = \lfloor \frac{\log n}{8[4+\log e+\log(\log n/f(n))]} \rfloor$ and $r = \lfloor (p-2)/(2f(n)) \rfloor$. We assume that n is large enough to ensure that $r \geq 1$. Consider the following algorithm for generating accesses for a tree T. If T is of height at least $\log^2 n/f(n)$, then locate the leftmost deepest node and access it r times. Otherwise, Lemma 3.8 implies that there exists a node x with turn sequence $\langle a_1, \ldots, a_k\rangle$ such that $k \geq p$ and $\sum_{i=1}^{k-p} b_i \geq \log n/4$, where b_i is the ith smallest element of the multiset $\{a_1, \ldots, a_k\}$; access x a total of r times. Note that, in either case, if at most p pointers are changed during r accesses, then the total access time is $\Omega(\frac{\log^2 n}{f(n)\log(\log n/f(n))})$.

Suppose an adversary applies the above algorithm I times to obtain the access sequence $S = \langle v_1\rangle^r\langle v_2\rangle^r \ldots \langle v_I\rangle^r$, where $I \geq \max(n, g(n))$. The splay heuristic takes only $O(I \log n + n \log n) = O(I \log n)$ time to access S. On the other hand, the heuristic H performs at most $g(n) + r \cdot I \cdot f(n) \leq p/2 \cdot I$ pointer changes when accessing S, which implies that H changes at most p pointers during at least $\lfloor I/2 \rfloor$ access sequences of the form $\langle v_j\rangle^r$. Hence, H requires at least $\Omega(I \cdot \frac{\log^2 n}{f(n)\log(\log n/f(n))})$ time to access S; therefore, it has a competitive ratio of $\Omega(\frac{\log n}{f(n)\log(\log n/f(n))})$. □

An unfortunate consequence of our lower bounds is that no $o(\log n)$-linkage-cost heuristic can have two desirable properties of the splay heuristic, the working set property, and the dynamic finger property, because either of the properties is sufficient to obtain the upper bounds on the splay heuristic used in our proofs. The working set property implies that a heuristic can quickly retrieve an element that was recently accessed. For splay trees, Sleator and Tarjan [21] proved that the amortized time

to access a sequence $\langle x_1, \ldots, x_m \rangle$ is $O(m + n \log n + \sum_{i=1}^{m} \log(t(i) + 1))$, where $t(i)$ is the number of distinct elements accessed between the ith access and the previous access of x_i, for $i = 1, \ldots, m$. The dynamic finger property implies that a heuristic can quickly access an element that is near the previously accessed element; for splay trees, Cole [10] showed that the amortized time to access $\langle x_1, \ldots, x_m \rangle$ is $O(m + n \log \log n + \sum_{i=1}^{m-1} \log(|k_{i+1} - k_i| + 1))$, where k_i is the symmetric-order position of x_i among the n tree elements for $i = 1, \ldots, m$.

4. Upper bounds. We now turn our attention to the problem of upper bounds. We present new heuristics that are based on the splay heuristic of Sleator and Tarjan [21]. In this section, we introduce three CLC heuristics and, in section 5, we present a practical, randomized, non-CLC heuristic. The splay heuristic two-promotes an accessed node until it reaches the root; this operation is called a *splay.* Sleator and Tarjan proved that the splay heuristic is statically competitive and conjectured that it is in fact competitive. Unfortunately, the splay heuristic has an *amortized linkage cost of* $\Theta(\log n)$. To achieve constant linkage cost, our heuristics perform only part of a splay. This approach, however, may not be adequate to achieve a low search cost, so we also use an extra restructuring step called a *local optimization* in two of the heuristics.

4.1. The randomized two-promotion heuristic (R2P). The first heuristic we present is a very simple one that supports only accesses. In the R2P heuristic, we attempt to perform $(2/n)$ths of the work of the splay heuristic. More precisely, suppose we access a node x on level l. We generate a random number L uniformly between 0 and $n - 1$, inclusive. If $L \leq l$, then we two-promote x's ancestor on level L. Otherwise, we do nothing. Clearly, the R2P heuristic is worst-case CLC, since we perform at most two promotions on each access. Also, it requires only constant extra space, and it can be implemented as a single top-down pass through the tree. Although the R2P heuristic is so simple, it achieves a surprisingly low expected access time, as we prove below.

Before analyzing the R2P heuristic, we briefly recall Sleator and Tarjan's amortized analysis of the splay heuristic [21]. They chose the potential $\Phi(T)$ of a tree T to be the total rank $R(T)$, and they analyzed the effect of a promotion and a two-promotion on $R(T)$.

LEMMA 4.1 (Sleator and Tarjan [21]). *The change in $R(T)$ after promoting a node x in a tree T is no greater than $3(r'(x) - r(x))$ if x is of depth of at least 1, and the change in $R(T)$ after two-promoting x is no greater than $3(r'(x) - r(x)) - 2$ if x is of depth at least 2, where $r'(x)$ is the rank of x after the restructuring operation.*

Sleator and Tarjan also showed that the change in $R(T)$ after splaying a node x on level l is no more than $3(r(t) - r(x)) - l + 1$, where t is the root of T.

LEMMA 4.2 (Sleator and Tarjan [21]). *Let t be the root of a tree T, and let x be a node on level l in T. The change in $R(T)$ after splaying x is no more than $3(r(t) - r(x)) - 2\lfloor l/2 \rfloor$.*

Since the time to access x is $l + 1$, the amortized time of splaying x is no more than $3(r(t) - r(x)) + 2$. By choosing node weights appropriately, we obtain various bounds. For example, if we assign a weight of 1 to each node, then the rank $r(t)$ of the root is $\log n$, so we obtain an $O(\log n)$ amortized access time.

Note that the splay heuristic two-promotes only half of the ancestors of an accessed node x. If we two-promote each ancestor of x with probability p, where the probabilities may or may not be independent, then we expect to do $2p$ths of the work

of the splay heuristic. We prove, in the following, that the expected change in $R(T)$ from this two-promotion scheme is essentially $2p$ times the change in $R(T)$ using the splay heuristic.

LEMMA 4.3. *Let t be the root of a tree T and x be a node of T on level l. Let x_i be the ancestor of x on level i for $i = 0, \ldots, l$. Suppose that, for $i = 0, \ldots, l$, the probability of each x_i being two-promoted is p such that for all j if x_j is two-promoted, then x_{j-1} and x_{j-2} are not. Then, the expected change ΔR in $R(T)$ is at most $p[6(r(t) - r(x)) - 2(l-1)]$.*

Proof. The lemma trivially holds if $l = 0$. If $l = 1$, then Lemma 4.1 implies that the expected change in potential is $\Delta R \leq p[3(r(x_0) - r(x_1))] < p[6(r(t) - r(x))]$. Finally, if $l \geq 2$, then Lemma 4.1 implies that

$$\begin{aligned}
\Delta R &\leq p[3(r(x_0) - r(x_1))] + p\sum_{i=2}^{l}[3(r(x_{i-2}) - r(x_i)) - 2] \\
&= p[3(r(x_0) - r(x_1))] + p[3(r(x_0) + r(x_1) - r(x_{l-1}) - r(x_l)) - 2(l-1)] \\
&< p[6(r(x_0) - r(x_l)) - 2(l-1)] \\
&= p[6(r(t) - r(x)) - 2(l-1)]. \qquad \square
\end{aligned}$$

Lemma 4.3 implies that the R2P heuristic adapts at a rate $1/(2p)$ as fast as the splay heuristic, yet also has an expected-amortized access time of $3(r(t) - r(x)) + 2$, if we choose the potential of a tree T to be $R(T)/(2p)$. Hence, in the following we analyze the running time of the R2P heuristic using the potential function $\Phi(T) = n/2 \cdot R(T)$.

We first bound the expected-amortized time of a single operation, before analyzing the expected time of a sequence of operations. We finally prove that the R2P heuristic is statically competitive.

LEMMA 4.4. *The expected-amortized time to access a node x of a tree with root t using the R2P heuristic is at most $3(r(t) - r(x)) + 2$.*

Proof. The expected-amortized time of an operation is the sum of the actual time and the expected change in potential. Let l be the depth of x. If $l = 0$, then the expected-amortized time to access x is $1 < 3(r(t) - r(x)) + 2$. Otherwise, the time to access x is $l + 1$, and since each ancestor of x is two-promoted with probability $1/n$, Lemma 4.1 implies that the expected change $\Delta\Phi$ in potential is no greater than

$$\begin{aligned}
\frac{n}{2}\Delta R &\leq \frac{n}{2}\left\{\frac{1}{n}[6(r(t) - r(x)) - 2(l-1)]\right\} \\
&= 3(r(t) - r(x)) - (l-1).
\end{aligned}$$

The expected-amortized time is thus at most $l + 1 + 3(r(t) - r(x)) - (l-1) = 3(r(t) - r(x)) + 2$. $\square$

We now bound the maximum expected time of a sequence of accesses by bounding the expected-amortized time of the sequence. Note that we have expected access frequencies in the following lemma, rather than observed access frequencies. The reason is that we assume the adversary is adaptive and maps histories of a tree to accesses in some manner. This mapping, along with the random numbers generated by the R2P heuristic, defines a probability distribution for the accesses, thus giving rise to expected access frequencies. For further details, readers should consult Lai's thesis [12].

LEMMA 4.5. *Let n be the size of a tree, m be the number of accesses, and q_i be the expected access frequency of the ith smallest element for $i = 1, \ldots, n$. Let w_i be*

an arbitrary, positive constant, for $i = 1, \ldots, n$, *and let* $W = \sum_{i=1}^{n} w_i$. *Then, the expected time of* m *accesses using the R2P heuristic is at most*

$$O(m) + 3 \cdot \sum_{i=1}^{n} \left[q_i \cdot \log\left(\frac{W}{w_i}\right)\right] + \frac{n}{2} \cdot \sum_{i=1}^{n} \log\left(\frac{W}{w_i}\right),$$

even if accesses are chosen by an adaptive adversary.

Proof. Let x_i be the ith smallest element, for each i. We assign x_i a weight of w_i; this implies that the root of the tree has a total weight of W. Recall that the expected time of m operations is no more than the sum of the expected-amortized time of the m operations and the maximum drop in potential.

Since the rank of the root of the tree is $\log W$ and the rank $r(x_i)$ of x_i is $\log s(x_i) \geq \log w_i$, the expected-amortized time of accessing x_i is at most $3\log(W/w_i)+2$. Therefore, the expected-amortized time of the m accesses is no more than

$$3\sum_{i=1}^{n} \left[q_i \cdot \log\left(\frac{W}{w_i}\right)\right] + 2m.$$

It remains to determine an upper bound on the maximum drop in potential. For each i, the rank of x_i is at least $\log w_i$ and at most $\log W$, which implies that the potential of the tree is at least $n/2 \cdot \sum_{i=1}^{n} \log w_i$ and at most $n/2 \cdot \sum_{i=1}^{n} \log W$. Hence, the potential drop is at most $n/2 \cdot \sum_{i=1}^{n} (\log W - \log w_i) = n/2 \cdot \sum_{i=1}^{n} \log(W/w_i)$. Therefore, the expected time E of the m accesses is at most

$$O(m) + 3 \cdot \sum_{i=1}^{n} \left[q_i \log\left(\frac{W}{w_i}\right)\right] + \frac{n}{2} \cdot \sum_{i=1}^{n} \log\left(\frac{W}{w_i}\right). \qquad \square$$

An amortized bound similar to that of Lemma 4.5 holds for splay trees, except that the term $n/2 \cdot \sum_{i=1}^{n} \log(W/w_i)$ is replaced by $\sum_{i=1}^{n} \log(W/w_i)$.

Note that Lemma 4.5 holds for all assignments of weights $w_1, \ldots, w_n$ simultaneously. By choosing different values of the weights, we obtain different upper bounds. Also, note that the specific values of the weights are unimportant; only the ratios between the weights affect the bound of Lemma 4.5. For example, if we choose all weights to be equal, we obtain the following bound.

THEOREM 4.6. *The expected time of* m *accesses using the R2P heuristic is* $O(m \log n + n^2 \log n)$, *even against an adaptive adversary.*

Proof. Assign a weight of $1/n$ to each item. Then $W = 1$, which implies that the expected-amortized access time is $O(\log n)$ and the maximum drop in potential is $O(n^2 \log n)$. The theorem follows. $\square$

Theorem 4.6 implies that the expected time of an access is essentially $O(\log n)$, even if an adversary is allowed to inspect the tree to choose accesses. However, the expected amount of *overwork* is $O(n^2 \log n)$, which is a factor of n greater than the overwork of the splay heuristic.

By choosing the weights differently, we prove that the R2P heuristic is statically competitive against an oblivious adversary.

THEOREM 4.7. *Let* n *be the size of a tree. Suppose that an oblivious adversary accesses a sequence* X *of length* m. *Let* q_i *be the access frequency of the* i*th smallest element in* X *for* $i = 1, \ldots, n$. *The expected time* T *of the* m *accesses using the R2P heuristic is*

$$O\left(m + n^2 \log n + \sum_{i=1}^{n} \left[q_i \log\left(\frac{m}{q_i}\right)\right]\right);$$

thus, R2P is statically competitive against an oblivious adversary.

Proof. Sherk [19], in his proof that deep-splay trees are statically optimal, proved that for any assignment of values to $q_1, \ldots, q_n$, there is an assignment to $w_1, \ldots, w_n$ such that

$$\sum_{i=1}^{n} \left[q_i \log \left(\frac{W}{w_i} \right) \right] = O\left(\sum_{i=1}^{n} \left[q_i \log \left(\frac{m}{q_i} \right) \right] \right),$$

and

$$\sum_{i=1}^{n} \log \left(\frac{W}{w_i} \right) = O(n \log n).$$

Therefore, Lemma 4.5 implies that the expected time of m accesses is

$$O\left(m + n^2 \log n + \sum_{i=1}^{n} \left[q_i \log \left(\frac{m}{q_i} \right) \right] \right).$$

Now, the adversary's access time T_A using an optimal static tree is $\Omega(m + \sum_{i=1}^{m} q_i \log(m/q_i))$, where m is the number of accesses and q_i is the access frequency of the ith smallest element. Since T is $O(T_A + n^2 \log n)$, R2P is statically competitive against an oblivious adversary. □

4.2. The locally optimized randomized partial-splay (LORPS) heuristic. Our first heuristic, the R2P heuristic, cannot support updates efficiently, since it rarely performs any restructuring if the height of the tree is low. The second heuristic, the LORPS heuristic, supports both accesses and updates. Furthermore, the LORPS heuristic is statically competitive in the sense that it is competitive against any static tree for any obliviously generated access sequence that does not contain intervening updates.

Let $d(n) = 2 + \log(n+1)$. The LORPS heuristic works as follows. After accessing some node x on level l, we generate a random number L uniformly between 0 and $\lfloor d(n) \rfloor - 1$. If $L > l$, we do nothing. Otherwise, we two-promote x's ancestors on levels in the set $\{i : i \leq l \text{ and } i \equiv L \pmod{\lfloor d(n) \rfloor}\}$. We call this operation a *partial splay,* since it performs $(2/\lfloor d(n) \rfloor)$ths or roughly $(2/\log n)$ths of the work of a splay.

To delete a node w with at most one child, we delete w and partially splay its parent. To delete an internal node x, we replace x by x's inorder successor y, delete y, and partially splay y's parent.

Inserting a node x on level l is more difficult. Let x_i be the ancestor of x on level i for $i = 0, \ldots, l$. Let k be the largest index such that $k \leq l$ and $s(x_k) \geq d(n)/\ln 2 + 1$. If there is no such k or if $l - k \leq 4$, we partially splay x. Otherwise, let

$$B_i = \frac{s(x_{i-2})[s(x_{i-2}) - s(x_{i+1})]}{s(x_{i+1})^2},$$

for all i, and let

$$C = \min_{k+3 \leq i \leq l-1} B_i.$$

If $C \geq 1$, we partially splay x. If $C < 1$, we two-promote some x_i such that $B_i = C$ and then partially splay x; we call the two-promotion of x_i a *local optimization.* Observe

that we do not need balance information since we can count the sizes of x_l, x_{l-1}, ... to obtain k. This computation requires only $O(d(n)) = O(\log n)$ time since we can stop immediately after we find an ancestor x_k such that $s(x_k) \geq d(n)/\ln 2 + 1$.

Intuitively, a local optimization is a type of balance operation. In particular, $\log B_i$ is an estimate of the change in total rank of the tree if x_i is two-promoted. Hence, a local optimization seeks to reduce the rank if the fringe or bottom of the tree is imbalanced. The reason for using local optimization is that a partial splay alone appears inadequate to achieve $O(\log n)$ expected insertion time in pathological situations. In particular, using our method of analysis, the expected amortized time of an insertion may be as high as $\Theta(\log n \log\log n)$. By using an extra two-promotion to improve the balance during insertions, we can provably achieve $O(\log n)$ expected operation time.

To analyze the LORPS heuristic, we first prove that the expected-amortized time of an access, an insertion, or a deletion is $O(\log n)$, before we analyze the expected time of a sequence of operations. In the following, we assign a weight of 1 to each node. To analyze the expected time, we choose the potential $\Phi_E(T)$ of a tree T to be $d(n)R(T)$. To analyze the expected number of two-promotions, we use the potential function $\Phi_{EP}(T) = R(T)$.

LEMMA 4.8. *The expected-amortized time to access x on level l in a tree, using the LORPS heuristic, is at most $8(r(t)-r(x))-l+3$, where t is the root of the tree. If each element has a weight of 1, the expected-amortized time is at most $8\log n - l + 3$. The expected-amortized number of two-promotions is no more than $7 - l/\lfloor d(n)\rfloor$.*

Proof. Recall that the expected-amortized time of an operation is the sum of the actual time and the expected change in potential. Each ancestor of x has a probability of $1/\lfloor d(n)\rfloor$ of being two-promoted, so Lemma 4.3 implies that the expected change $\Delta\Phi$ in potential is less than

$$\frac{d(n)}{\lfloor d(n)\rfloor}[6(r(t) - r(x)) - 2(l-1)].$$

Since $n \geq 1$, we have $1 \leq d(n)/\lfloor d(n)\rfloor < 4/3$, which implies that

$$\Delta\Phi < 8(r(t) - r(x)) - 2(l-1).$$

Thus, the expected-amortized time is at most $l + 1 + 8(r(t) - r(x)) - 2(l-1) = 8(r(t)-r(x)) - l + 3$. Since the rank of the root t is $\log n$ and the rank of x is at least 0, the expected-amortized time is at most $8\log n - l + 3$.

Similarly, it is straightforward to show that the expected number of two-promotions is $l/\lfloor d(n)\rfloor$, and the expected change in the potential function Φ_{EP} is at most $6/\lfloor d(n)\rfloor \log n - (2(l-1))/\lfloor d(n)\rfloor$, which implies that the expected-amortized number of two-promotions is at most $(6/\lfloor d(n)\rfloor)\log n - l/\lfloor d(n)\rfloor + 1 < 7 - l/\lfloor d(n)\rfloor$. □

Before analyzing the expected-amortized time of an insertion, we establish an upper bound on the change in potential caused by a local optimization. Intuitively, Lemma 4.9 states that a local optimization negates most of the increase in potential caused by increases in ranks along the access or insertion path.

LEMMA 4.9. *Let $d(n)$ be some nondecreasing function of the tree size n. Let $x_0, \ldots, x_l$ be the nodes of the path from the root to x_l in T. Assume that $s(x_0) \geq d(n)/\ln 2 + 1$. Let B_i, for $k+3 \leq i \leq i-1$, and C be defined as above, where k is the largest index such that $k \leq l$ and $s(x_k) \geq d(n)/\ln 2 + 1$. Suppose that we increase the size of x_i by 1, for $i = 0, \ldots l-1$, and that we do the following steps.*

1. *If $k > l - 5$ or $C \geq 1$, do nothing.*

2. *Otherwise, two-promote some x_a such that $B_a = C$.*

Then, the total change in $\Phi(T) = d(n)R(T)$ caused by the increase in total rank and the restructuring is less than $l + \frac{15}{2}d(n)$.

Proof. The change in potential $\Phi(T)$ is caused by the change in the sizes of x_0, $\ldots$, x_{l-1} and the two-promotion of x_a if any. The change in potential caused by the change in the sizes is exactly $d(n) \cdot \sum_{i=0}^{l-1} \log(\frac{s(x_i)}{s(x_i)-1})$, where $s(x_i)$ is the size of x_i after the change in weight but before the extra two-promotion of x_a if any. Since $\ln(1+y) \le y/\ln 2$, for any $y \ge 0$, we have

$$\begin{aligned} d(n) \cdot \sum_{i=0}^{l-1} \log\left(\frac{s(x_i)}{s(x_i)-1}\right) &\le d(n) \cdot \sum_{i=0}^{k} \log\left(\frac{s(x_i)}{s(x_i)-1}\right) + d(n) \cdot \sum_{i=k+1}^{l-1} \log\left(\frac{s(x_i)}{s(x_i)-1}\right) \\ &\le \frac{d(n)}{\ln 2} \sum_{i=0}^{k} \frac{1}{s(x_i)-1} + d(n) \cdot \sum_{i=k+1}^{l-1} \log\left(\frac{s(x_i)}{s(x_i)-1}\right) \\ &\le k+1 + d(n) \cdot \sum_{i=k+1}^{l-1} \log\left(\frac{s(x_i)}{s(x_i)-1}\right) \end{aligned}$$

by the definition of k.

We claim that the total change in Φ is less than $l + \frac{15}{2}d(n)$. We consider the three possibilities separately.

1. $k \ge l-4$.
   If $k \ge l-4$, then no local optimization is performed. Since $s(x_i) \ge 2$ for $i < l$, we know that
   $$d(n) \cdot \sum_{i=k+1}^{l-1} \log\left(\frac{s(x_i)}{s(x_i)-1}\right) < 3d(n).$$
   Thus, the change in potential is at most $k+1+3d(n) < l+4d(n)$.
2. $k \le l-5$ and $C \ge 1$.
   Let r be the largest index such that $r \le l-1$ and $s(x_r) \ge 5$; such an r must exist since $l \ge 5$, which implies that $n \ge 6$. We have $\sum_{i=r+1}^{l-1} \log(\frac{s(x_i)}{s(x_i)-1}) \le \log 4 = 2$, which implies that
   $$d(n) \cdot \sum_{i=k+1}^{l-1} \log\left(\frac{s(x_i)}{s(x_i)-1}\right) \le \frac{d(n)}{\ln 2} \sum_{i=k+1}^{r} \frac{1}{s(x_i)-1} + 2d(n).$$
   To bound $\sum_{i=k+1}^{r} \frac{1}{s(x_i)-1}$, we show that the sequence $s(x_r)$, $s(x_{r-1})$, $\ldots$, increases exponentially. For any $k+3 \le i \le l-1$, we know by the definition of C that
   $$C \le \frac{s(x_{i-2})[s(x_{i-2}) - s(x_{i+1})]}{s(x_{i+1})^2}.$$
   Hence, $s(x_{i-2})^2 - s(x_{i+1})s(x_{i-2}) - C \cdot s(x_{i+1})^2 \ge 0$, which implies that
   $$\begin{aligned} s(x_{i-2}) &\ge \frac{s(x_{i+1}) + \sqrt{s(x_{i+1})^2 + 4C \cdot s(x_{i+1})^2}}{2} \\ &= \left(\frac{1+\sqrt{1+4C}}{2}\right) s(x_{i+1}). \end{aligned}$$

Since $C \geq 1$, we know that $s(x_{i-2}) \geq (\frac{1+\sqrt{5}}{2})s(x_{i+1})$. By induction, we can show that, for all $j \geq 0$, $s(x_{r-j}) \geq 5(\frac{1+\sqrt{5}}{2})^{\lfloor j/3 \rfloor}$. Thus,

$$\begin{aligned}\frac{d(n)}{\ln 2}\sum_{i=k+1}^{r}\frac{1}{s(x_i)-1} &< \frac{d(n)}{\ln 2}\sum_{j=0}^{\infty}\left[\frac{1}{4(\frac{1+\sqrt{5}}{2})^{\lfloor j/3 \rfloor}}\right] \\ &= \frac{d(n)}{\ln 2}\cdot\frac{3}{4}\cdot\sum_{j=0}^{\infty}\frac{1}{(\frac{1+\sqrt{5}}{2})^{j}} \\ &\leq \frac{d(n)}{\ln 2}\cdot\frac{3}{4}\cdot\frac{\sqrt{5}+1}{\sqrt{5}-1} \\ &< 3d(n).\end{aligned}$$

Therefore, the change in the potential from the changes in the sizes of x_{k+1}, $\ldots$, x_{l-1} is less than $3d(n) + 2d(n) = 5d(n)$. The total change in potential is less than $k + 1 + 5d(n) < l + 5d(n)$.

3. $k \leq l - 5$ and $C < 1$.
Let r be the largest index such that $r \leq l - 1$ and $s(x_r) \geq \lceil 2/C \rceil + 1$. If no such r exists, then

$$\sum_{i=k+1}^{l-1}\log\left(\frac{s(x_i)}{s(x_i)-1}\right) \leq d(n)\log\left\lceil\frac{2}{C}\right\rceil.$$

Otherwise, $\sum_{i=r+1}^{l-1}\log(\frac{s(x_i)}{s(x_i)-1}) \leq \log s(x_{r+1}) \leq \log\lceil 2/C \rceil$, which implies that

$$d(n)\cdot\sum_{i=k+1}^{l-1}\log\left(\frac{s(x_i)}{s(x_i)-1}\right) \leq \frac{d(n)}{\ln 2}\sum_{i=k+1}^{r}\frac{1}{s(x_i)-1} + d(n)\log\left\lceil\frac{2}{C}\right\rceil.$$

To bound $\sum_{i=k+1}^{r}\frac{1}{s(x_i)-1}$, we analyze the rate of growth of the sequence $s(x_r)$, $s(x_{r-1})$, $\ldots$. Let $z = \lceil 2/C \rceil/(2/C)$. Since $C < 1$, we have $1 \leq z < 3/2$. Observe that, for all $k + 3 \leq i \leq l - 1$, we have

$$\begin{aligned}s(x_{i-2}) &\geq \left(\frac{1+\sqrt{1+4C}}{2}\right)s(x_{i+1}) \\ &> \left(1+\frac{C}{2}\right)s(x_{i+1}).\end{aligned}$$

We can show by induction that, for $j \geq 0$, $s(x_{r-j}) - 1 > (1 + C/2)^{\lfloor j/3 \rfloor}\lceil 2/C \rceil$. Thus,

$$\begin{aligned}\frac{d(n)}{\ln 2}\sum_{i=k+1}^{r}\frac{1}{s(x_i)-1} &< \frac{d(n)}{\ln 2}\left(\frac{3}{\lceil\frac{2}{C}\rceil}\right)\sum_{i=0}^{\infty}\frac{1}{(1+\frac{C}{2})^{i}} \\ &= \frac{d(n)}{\ln 2}\left(\frac{3}{\lceil\frac{2}{C}\rceil}\right)\left(\frac{1+\frac{C}{2}}{\frac{C}{2}}\right) \\ &= \frac{d(n)}{\ln 2}\cdot\frac{3(1+\frac{C}{2})}{z} \\ &< \frac{9d(n)}{2z\ln 2}.\end{aligned}$$

Therefore,

$$d(n) \cdot \sum_{i=k+1}^{l-1} \log\left(\frac{s(x_i)}{s(x_i)-1}\right) < d(n)\left[\frac{9}{2z\ln 2} + \log\left(\frac{2}{C}\right) + \log z\right].$$

Now, $9/(2z\ln 2) + \log z$ is maximized when $z = 1$, so the change in potential due to the change in ranks of $x_{k+1}, \ldots, x_{l-1}$ is

$$d(n) \cdot \sum_{i=k+1}^{l-1} \log\left(\frac{s(x_i)}{s(x_i)-1}\right) \le d(n)\left[\frac{9}{2\ln 2} + 1 - \log C\right].$$

Let x_a be the node that has been two-promoted. Only the ranks of x_{a-2}, x_{a-1}, and x_a are changed. Before the two-promotion, the sum of their ranks is $\log s(x_{a-2}) + \log s(x_{a-1}) + \log s(x_a) > \log s(x_{a-2}) + 2\log s(x_{a+1})$. After the two-promotion, the size of either x_{a-1} or x_{a-2} can be shown to be less than $s(x_{a-2}) - s(x_{a+1})$, while the size of the other node is clearly less than $s(x_{a-2})$. Thus, the sum of the new ranks of x_{a-2}, x_{a-1}, and x_a is no greater than $2\log s(x_{a-2}) + \log(s(x_{a-2}) - s(x_{a+1}))$. The change $\Delta\Phi$ in potential from the local optimization is less than

$$d(n)[\log s(x_{a-2}) + \log(s(x_{a-2}) - s(x_{a+1})) - 2\log s(x_{a+1})]$$

and, hence,

$$\begin{aligned}\Delta\Phi &< d(n)\left\{\log\frac{s(x_{a-2})[s(x_{a-2}) - s(x_{a+1})]}{s(x_{a+1})^2}\right\} \\ &= d(n)\log C.\end{aligned}$$

Thus, the sum of the change in $d(n) \cdot \sum_{i=k+1}^{l-1} \log(\frac{s(x_i)}{s(x_i)-1})$ and the change in potential caused by the local optimization is at most $(\frac{9}{2\ln 2}+1)d(n) < \frac{15}{2}d(n)$. The total change in potential is, therefore, less than $l + \frac{15}{2}d(n)$. The lemma follows immediately. □

Now that we have analyzed the effect of a local optimization, we bound the expected-amortized time for an insertion, before analyzing the time for a deletion.

LEMMA 4.10. *The expected-amortized insertion time using the LORPS heuristic is $O(\log n)$, and the expected-amortized number of two-promotions is constant.*

Proof. Clearly, for $n \le 8$, the expected-amortized time is $O(1)$. If $n > 8$, then the expected-amortized time of inserting a node x is the sum of the search time, the change in potential caused by the change in $d(n)$, the change in potential caused by the addition of x in various subtrees, the expected-amortized time of partially splaying x, and the expected change in potential caused by the local optimization (if any).

Lemma 4.8 implies that the sum of the expected-amortized time for searching the tree for x and partially splaying x is $l + 1 + 8\log n - 2(l-2) + 2 = 8\log n - l + 7$, where l is the depth at which x is originally inserted. The reason we have $-2(l-2)$ instead of $-2l$ is that x may be moved upward two levels during the local optimization before it is partially splayed.

The change in potential caused by the change in $d(n)$ is at most $(d(n) - d(n-1))n\log n$. Observe that

$$d(n) - d(n-1) = \log\left(1 + \frac{1}{n}\right).$$

We can show that $\log(1+1/n) \le 1/(n \ln 2)$, so the change in potential caused by the change in $d(n)$ is at most $(n \log n)/(n \ln 2) = (\log n)/\ln 2$.

Let x_i be the ancestor of x on level l for $0 \le i \le l$. Let k be the largest index such that $k \le l$ and $s(x_k) \ge d(n)/\ln 2 + 1$; such a k must exist if $n \ge 9$. Lemma 4.9 implies that the change in potential from the change in sizes and the local optimization, if any, is at most $l + \frac{15}{2} d(n) = l + O(\log n)$. Therefore, the expected-amortized time of inserting x is $O(\log n)$.

The expected number of two-promotions performed when inserting x is at most $l/\lfloor d(n) \rfloor + 1$, and the preceding proof implies that the change in the potential function Φ_{EP} is $O(\log n/\lfloor d(n) \rfloor) - l/\lfloor d(n) \rfloor$. Therefore, the expected-amortized number of two-promotions is $O(\log n/\lfloor d(n) \rfloor) = O(1)$. □

LEMMA 4.11. *The expected-amortized deletion time using the LORPS heuristic is $O(\log n)$, and the expected-amortized number of two-promotions is constant.*

Proof. Without loss of generality assume that the deleted node x has at most one child. The expected-amortized time to delete x is the sum of the time to search for x, the change in potential from the change in $d(n)$, the change in the potential from removing x from various subtrees, and the expected-amortized time of partially splaying x's parent. A deletion causes $d(n)$ and the ranks of x's ancestors to decrease, which implies that the change in potential is negative. Lemma 4.8 implies that the expected-amortized time to search for x, remove it, and partially splay its parent is at most $l + 1 + 8 \log n - 2(l-1) + 2 \le 8 \log n + 5$, where l is the original depth of x.

The expected number of two-promotions performed when deleting x is less than $l/\lfloor d(n) \rfloor$, and the above proof implies that the change in the potential function Φ_{EP} is $O(\log n/\lfloor d(n) \rfloor) - l/\lfloor d(n) \rfloor$. Therefore, the expected-amortized number of two-promotions is $O(\log n/\lfloor d(n) \rfloor) = O(1)$. □

We are now ready to show that the LORPS heuristic effectively has an expected-amortized update time of $O(\log n)$.

THEOREM 4.12. *Suppose that we perform m operations on an empty tree using the LORPS heuristic. Let p_{in} be the probability that the size of the tree is n after the ith operation. The expected time of m operations is*

$$O\left(m + \sum_{i=1}^{m} \sum_{j=0}^{i} [p_{ij} \log(j+1)]\right)$$

and the expected linkage cost of m operations is $O(m)$, even if operations are chosen by an adaptive adversary. Essentially, the expected cost of an operation is $O(\log n)$, and the expected linkage cost is constant.

Proof. Recall that the expected time is no more than the sum of the expected-amortized time and the maximum drop in potential. Observe that $\Phi_E = 0$ and $\Phi_{EP} = 0$ when the tree is empty, so the maximum potential drop is 0. The expected time bound immediately follows from Lemmas 4.8, 4.10, and 4.11. Since Lemmas 4.8, 4.10, and 4.11 also imply that the expected-amortized number of two-promotions, for any operation, is constant, the expected linkage cost of the m operations is $O(m)$. □

Theorem 4.12 implies that the LORPS heuristic is as efficient as any balanced tree. Although it is not practical and requires restructuring during accesses, unlike red–black trees [11] or randomized search trees [3], it has the advantage of requiring no balance information to achieve $O(\log n)$ expected operation cost.

By varying the node weights as in our analysis of the R2P heuristic, we show that the LORPS heuristic is also statically competitive.

THEOREM 4.13. *Let n be the size of a tree. Suppose an oblivious adversary accesses a sequence X of length m. Let q_i be the access frequency of the ith smallest element in X for $i = 1, \ldots, n$. The expected time of the m searches using the LORPS heuristic is*

$$O\left(m + n\log^2 n + \sum_{i=1}^{n}\left[q_i \log\left(\frac{m}{q_i}\right)\right]\right);$$

thus, LORPS is statically competitive against an oblivious adversary.

Proof. Let x_i be the ith smallest element for each i. We assign x_i a weight of w_i such that

$$\sum_{i=1}^{n}\left[q_i \log\left(\frac{W}{w_i}\right)\right] = O\left(\sum_{i=1}^{n}\left[q_i \log\left(\frac{m}{q_i}\right)\right]\right)$$

and

$$\sum_{i=1}^{n}\log\left(\frac{W}{w_i}\right) = O(n\log n),$$

where $W = \sum_{i=1}^{n} w_i$; Sherk [19] proved that this assignment is possible for any values of $q_1, \ldots, q_n$.

Lemma 4.8 implies that the expected-amortized time of accessing x_i is

$$8(r(t) - r(x_i)) + 3 \le 8\log(W/w_i) + 3,$$

where $r(t)$ is the rank of the root of the tree. Thus, the expected-amortized time of m accesses is at most $O(m) + 8\sum_{i=1}^{n}[q_i \log(W/w_i)] = O(m + \sum_{i=1}^{n}[q_i \log(m/q_i)])$. For each i, the rank of x_i is at least $\log w_i$ and at most $\log W$, which implies that the maximum potential drop is at most $d(n) \cdot \sum_{i=1}^{n}\log(W/w_i) \le O(d(n) \cdot n\log n) = O(n\log^2 n)$. The theorem follows in a similar manner to the proof of Theorem 4.7. □

4.3. The locally optimized partial-splay (LOPS) heuristic. The third heuristic, the LOPS heuristic, is a deterministic, statically competitive, amortized-CLC heuristic. The LOPS heuristic is essentially a deterministic version of the LORPS heuristic that maintains bounded frequency counts. The local optimizations play the same role in LOPS as they do in LORPS; namely, they are rebalancing operations. The LOPS heuristic achieves amortized time bounds comparable with the LORPS heuristic's expected-time bound, although at the expense of linear extra space.

At first sight, it appears that we can use Ben-David et al.'s results [5] to obtain a deterministic algorithm from the LORPS heuristic. They provide a general technique, essentially maintaining an estimate of the potential, to convert a online randomized algorithm into a online deterministic algorithm with little loss in competitiveness. The main problem with their approach is that it does not handle static competitiveness and whether it could be modified to handle static competitiveness is unclear.

In the LOPS heuristic we store a positive integer weight in each node. Let $d(n) = 2 + \log(n+1)$. Suppose we access x on level l. There are two steps.

1. Increase the weight of x by 1. Let x_i be the ancestor of x on level l for $i = 0, \ldots, l$. Let k be the largest index such that $k \le l$ and $s(x_k) \ge d(n)/\ln 2 + 1$. If no such k exists or $l - k \le 4$, go to step 2. Otherwise, let

$$B_i = \frac{s(x_{i-2})[s(x_{i-2}) - s(x_{i+1})]}{s(x_{i+1})^2},$$

for all i, and let

$$C = \min_{k+3 \le i \le l-1} B_i.$$

If $C \ge 1$, go to step 2. Otherwise, two-promote some x_i such that $B_i = C$. We call step 1 a *local optimization*.

2. Let l' be the new level of x and let x'_i be the ancestor of x on level i for $i = 0, \ldots, l'$. If $l' = 0$, we are done. Otherwise, let $D_j = \lfloor (l' - j)/\lfloor d(n) \rfloor \rfloor$ for $j = 1, \ldots, \lfloor d(n) \rfloor$. Let $\delta_i = 3(r(x'_{i-2}) - r(x'_i)) - 2$, for $i = 2, \ldots, l'$. Compute

$$R_1 = 3(r(x'_0) - r(x'_1)) + \sum_{j=1}^{D_1} \delta_{\lfloor d(n) \rfloor j+1},$$

and, for $i = 2, \ldots, \lfloor d(n) \rfloor$,

$$R_i = \sum_{j=0}^{D_i} \delta_{\lfloor d(n) \rfloor j+i}.$$

Let $C' = \min(R_1, \ldots, R_{\lfloor d(n) \rfloor})$. If $C' \ge 0$, then we are done. Otherwise, two-promote x's ancestors on levels in $\{i : i \le l \text{ and } i \equiv L \pmod{\lfloor d(n) \rfloor}\}$ for some L such that $R_L = C'$. We call step 2 a *partial splay*.

Updates are straightforward. To delete a node with at most one child, we delete the node and partially splay its parent. To delete an internal node x, we replace x by x's inorder successor y, delete y, and partially splay y's parent. To insert some node x, we assign x a weight of 0 and access x, applying both steps 1 and 2.

To reduce the size of the weights, we reset the weights to 1 after every $\lfloor n/2 \rfloor + 1$ operations. This step ensures that we need only $O(n)$ extra space or $O(n \log n)$ bits of extra space in a bit-cost model.

The analysis of the LOPS heuristic is similar to that of the LORPS heuristic. To analyze the amortized time of the LOPS heuristic, we use the potential function $\Phi_A(T) = d(n) \cdot R(T)$. To analyze the amortized number of two-promotions, we use the potential function $\Phi_{2P}(T) = R(T)$. We note that the amortized cost of resetting weights is constant, so we ignore it in the analysis.

We first analyze the amortized costs of an access, an insertion, and a deletion before analyzing the amortized cost of a sequence of operations.

LEMMA 4.14. *The amortized time to access node x of a tree with root t using the LOPS heuristic is $O(\log(s(t)/s(x)))$. The amortized number of two-promotions performed when accessing x is constant.*

Proof. Let l be the depth of x. Without loss of generality, assume that $n \ge 9$ since the amortized access time is clearly constant otherwise. The amortized time to access x is the sum of the actual time, which is $l + 1$, and the change in potential, caused by the increases in sizes, the local optimization (if any), the partial splay, and the resetting of weights (if any). We may ignore the contribution of the weight resetting since it reduces the potential.

Let $\Delta\Phi$ be the change in potential Φ_A caused by the increases in sizes and the local optimization. For $0 \le i \le l$, let x_i be the ancestor of x on level i before the

access. Let k be the largest index such that $k \le l$ and $s(x_k) \ge d(n)/\ln 2 + 1$; such a k exists if $n \ge 9$. If $k = l$, then no local optimization is performed and

$$\begin{aligned}\Delta\Phi &= d(n) \cdot \sum_{i=0}^{l} \log\left(\frac{s(x_i)}{s(x_i)-1}\right) \\ &\le \frac{d(n)}{\ln 2} \sum_{i=0}^{l} \frac{1}{s(x_i)-1} \\ &\le l+1.\end{aligned}$$

If $k \ne l$, then Lemma 4.9 implies that the change $\Delta\Phi$ in potential is

$$d(n) \cdot \log\left(\frac{s(x_l)}{s(x_l)-1}\right) + l + \frac{15}{2}d(n) \le l + \frac{17}{2}d(n).$$

Let $\Delta\Phi'$ be the change in potential caused by the partial splay. Let l' be the level of x after step 1 of the access algorithm and, for all i, $0 \le i \le l'$, let x'_i be the ancestor of x on level i after step 1. Observe that Lemma 4.1 implies that the change in potential after two-promoting x's ancestors on levels in $\{i : i \le l \text{ and } i \equiv L \pmod{\lfloor d(n) \rfloor}\}$ is at most $d(n) \cdot R_L$. Thus, the change $\Delta\Phi'$ in potential from the partial splay is at most $d(n) \cdot \min(R_1, \ldots, R_{\lfloor d(n) \rfloor}, 0)$. Clearly,

$$\min(R_1, \ldots, R_{\lfloor d(n) \rfloor}) \le \frac{R_1 + \cdots + R_{\lfloor d(n) \rfloor}}{\lfloor d(n) \rfloor}.$$

If $l' \ge 2$, then, since $l' \ge l - 2$ and $d(n) \ge 3$, we know that

$$\begin{aligned}\Delta\Phi' &\le \frac{d(n)}{\lfloor d(n) \rfloor} \sum_{i=1}^{\lfloor d(n) \rfloor} R_i \\ &= \frac{d(n)}{\lfloor d(n) \rfloor} \cdot 3(r(x'_0) - r(x'_1)) + \frac{d(n)}{\lfloor d(n) \rfloor} \sum_{i=2}^{l'} \left[3(r(x'_{i-2}) - r(x'_i)) - 2\right] \\ &= \frac{d(n)}{\lfloor d(n) \rfloor}[6r(x'_0) - 3r(x'_{l-1}) - 3r(x'_l) - 2(l'-1)] \\ &< 8r(x'_0) - 8r(x'_l) - 2l' + 2 \\ &\le 8r(t) - 8r(x) - 2l + 6,\end{aligned}$$

where $r(t)$ is the rank of the root after the increases in sizes. Note that the above bound also holds if $l' < 2$.

Therefore, the amortized access time A is at most

$$\begin{aligned}l + 1 + \Delta\Phi + \Delta\Phi' &< 8[r(t) - r(x)] - l + 7 + \Delta\Phi \\ &= O(\log(s(t)/s(x))) - l + \Delta\Phi.\end{aligned}$$

If $s(x) \ge d(n)/\ln 2 + 1$, then $k = l$ and the amortized access time A is $O(\log(s(t)/s(x))) - l + l + 1 = O(\log(s(t)/s(x)))$. Otherwise, if $s(x) < d(n)/\ln 2 + 1$, then $s(t) \ge n$ implies that $\log(s(t)/s(x)) = \Omega(\log n)$ and $A = O(\log(s(t)/s(x))) - l + l + \frac{17}{2}d(n) = O(\log(s(t)/s(x)))$.

The amortized number of two-promotions performed is at most $l/\lfloor d(n) \rfloor + 2 + (\Delta\Phi + \Delta\Phi')/d(n) = O(\log(s(t)/s(x))/\lfloor d(n) \rfloor)$. Since we reset all weights to 1 every

$\lceil n/2 \rceil$ operations, we know that $s(t) = O(n^2)$, which implies that the amortized number of two-promotions is constant. □

LEMMA 4.15. *The amortized insertion time using the LOPS heuristic is* $O(\log n)$, *and the amortized number of two-promotions is constant.*

Proof. The amortized time to insert a node x is the sum of the actual time and the change in potential, or, equivalently, the sum of the change in potential caused by the change in $d(n)$ and the amortized time to access x. Since all weights are at most n, the change $\Delta\Phi$ in potential from the change in $d(n)$ is

$$\begin{aligned} O((d(n) - d(n-1))n \log n) &= O((\log(n+1) - \log n)n \log n) \\ &= O(\log n). \end{aligned}$$

Lemma 4.14 implies that the amortized access time of x is $O(\log n)$, so the total amortized time is $O(\log n)$.

The amortized number of two-promotions to insert x is equal to the amortized number of two-promotions to access x, which is constant. □

LEMMA 4.16. *The amortized deletion time using the LOPS heuristic is* $O(\log n)$, *and the amortized number of two-promotions is constant.*

Proof. The amortized time to delete a node x on level l is the sum of the actual time, which is $l + 1$, and the change in potential caused by the change in $d(n)$, the change in sizes, and the partial splay. Since $d(n)$ and the sizes decrease, it is sufficient to consider the change in potential $\Delta\Phi_A$ from the partial splay.

It is sufficient to consider the case where x has at most one child. If x is the root, then the amortized deletion time is clearly $O(1)$. Otherwise, let y be x's parent. The proof of Lemma 4.14 implies that $\Delta\Phi_A < 8(r(t) - r(y)) - 2(l-1) + 2$, where $r(t)$ is the rank of the root. (We have $-2(l-1)$ instead of $-2l$ since we partially splay y, which is on level $l-1$.) Thus, the amortized deletion time is at most $l+1+8(r(t)-r(y))-2l+4 = O(\log n)$.

The amortized number of two-promotions to delete x is the sum of the actual number of two-promotions, which is less than $l/\lfloor d(n) \rfloor + 1$, and the change in potential $\Delta\Phi_{2P}$. The preceding upper bound on $\Delta\Phi_A$ implies that $\Delta\Phi_{2P} \leq c - l/d(n)$, for some constant c, which implies that the amortized number of two-promotions is constant. □

Before proving our main theorem on the time of the LOPS heuristic, we state a useful relationship involving logarithms. Note that we define $0 \log(x/0) = 0$.

LEMMA 4.17. *For any* $0 \leq a \leq x$ *and* $0 \leq b \leq y$, $a \log(x/a) + b \log(y/b) \leq (a+b) \log((x+y)/(a+b))$.

We are now ready to prove our main result: the LOPS heuristic supports all operations in $O(\log n)$ amortized time, is amortized CLC, and is statically competitive.

THEOREM 4.18. *Suppose we perform* m *operations in an empty tree using the LOPS heuristic. Let* n_i *be the size of the tree during the* i*th operation. Then, the worst-case time of the* m *operations is*

$$O\left(m + \sum_{i=1}^{m} \log(n_i + 1)\right).$$

The worst-case linkage cost of the m *operations is* $O(m)$.

Proof. Observe that the potential functions Φ_A and Φ_{2P} are initially 0, which implies that the maximum potential drop is 0 and that the worst-case time of the m

operations is no greater than the sum of the amortized times of the operations. The proof follows from the amortized time bounds of Lemmas 4.14, 4.15, and 4.16. ∎

Since LOPS is a deterministic heuristic, adaptive and oblivious adversaries are equally powerful.

THEOREM 4.19. *Let n be the size of a tree. Suppose that an adversary accesses a sequence X of length m. Let q_i be the access frequency of the ith smallest element in X for $i = 1, \ldots, n$. The worst-case time of the m searches using the LOPS heuristic is*

$$O\left(m + n\log^2 n + \sum_{i=1}^{n}\left[q_i \log\left(\frac{m}{q_i}\right)\right]\right);$$

thus, LOPS is statically competitive.

Proof. Since the maximum weight in the tree in no more than n, the maximum rank of the tree and hence the maximum drop in potential Φ_A is $O(n\log^2 n)$. Thus, it is sufficient to show that the sum of the amortized times of the m accesses is $O(m + n\log^2 n + \sum_{i=1}^{n}(q_i \log(m/q_i)))$. Clearly, Lemma 4.14 implies this bound if $m \le n$, so we assume in the following that $m > n$.

For $i = 1, \ldots, n$, let x_i be the ith smallest element in X, and let T_i be the total amortized time of the q_i accesses of x_i. To prove the claimed time bound, it is sufficient to show that $T_i = O(q_i + q_i \log(m/q_i))$. Let R be the number of times that the weights have been reset. Let Q_{ij} be the number of times that x_i is accessed between the jth and $(j+1)$th weight resettings for $0 \le j \le R$. Since we reset the weights after every $\lfloor n/2 \rfloor + 1$ operations, we know that the total weight of the tree before the $(j+1)$th weight resetting is at most $\lfloor 3n/2 \rfloor + 1$. Thus, we have

$$\begin{aligned}
T_i &= O\left(q_i + \sum_{j=0}^{R}\sum_{k=1}^{Q_{ij}} \log\left(\frac{\lfloor \frac{3}{2}n \rfloor + 1}{k}\right)\right) \\
&= O\left(q_i + \sum_{j=0}^{R}\left[Q_{ij}\log\left(\left\lfloor \frac{3n}{2}\right\rfloor + 1\right) - \log Q_{ij}!\right]\right) \\
&= O\left(q_i + \sum_{j=0}^{R}\left[Q_{ij}\log\left(\left\lfloor \frac{3n}{2}\right\rfloor + 1\right) - Q_{ij}\log Q_{ij} + \frac{Q_{ij}}{\ln 2}\right]\right) \\
&= O\left(q_i + \sum_{j=0}^{R}\left[Q_{ij}\log\frac{\lfloor \frac{3}{2}n \rfloor + 1}{Q_{ij}}\right]\right).
\end{aligned}$$

Because we reset weights every $\lfloor n/2 \rfloor + 1$ operations, the fact that we have performed m operations implies that $\sum_{j=0}^{R-1}(\lfloor n/2 \rfloor + 1) \le m$. Also, we know that $n < m$, so

$$\sum_{j=0}^{R}\left[\left\lfloor \frac{3n}{2}\right\rfloor + 1\right] \le \frac{9}{2}m + 1.$$

Since $\sum_{j=0}^{R} Q_{ij} = q_i$, Lemma 4.17 implies that

$$\begin{aligned}
T_i &= O\left(q_i + q_i \log\left(\frac{\frac{9}{2}m + 1}{q_i}\right)\right) \\
&= O\left(q_i + q_i \log\left(\frac{m}{q_i}\right)\right).
\end{aligned}$$

The theorem follows in a similar manner to the proof of Theorem 4.7. □

5. The randomized partial-splay (RPS) heuristic. The last heuristic we present, the RPS heuristic, performs $(2/d)$ths of the work of the splay heuristic. By adjusting the parameter d, one can obtain a tradeoff between the rate of adaptation and the linkage cost. More precisely, the RPS heuristic works as follows. Let $d \geq 3$ be an integer constant. After accessing some node x on level l, we generate a random number L uniformly between 0 and $d-1$. If $L > l$, we do nothing. Otherwise, we two-promote x's ancestors on levels in the set $\{i : i \leq l \text{ and } i \equiv L \pmod{d}\}$. We note, as an aside, that Sleator and Tarjan's semisplay step [21] can be used in place of the two-promotion.

Updates are straightforward. To delete a node with at most one child, we just delete the node and partially splay its parent. To delete an internal node x, we replace x by x's inorder successor y, delete y, and partially splay y's parent. To insert some node x, we insert x naively and then partially splay x.

Although the RPS heuristic is not CLC, it is practical and is much simpler to implement than the LORPS and LOPS heuristics. Because the RPS heuristic spaces out its two-promotions, the two-promotions do not interact with one another, so the RPS heuristic has a straightforward top-down implementation. We show that the RPS heuristic is statically competitive and has performance bounds similar to those of the splay heuristic. The RPS heuristic does not adapt as quickly as the splay heuristic to changes in usage but may be preferable to the splay heuristic if updates are infrequent and the access pattern is stable.

We first prove a technical result that bounds the change in potential caused by a partial splay.

LEMMA 5.1. *Suppose, for some node x, we two-promote x's ancestors on levels $l(1), \ldots, l(k)$, such that $l(i+1) \geq l(i) + 3$, for any i. Then, the change in $R(T)$ in the worst case is at most $3(r(t) - r(x_{l(k)})) - 2(k-1)$, where t is the root of the tree.*

Proof. Let x_i be the ancestor of x on level i for all i. For convenience, we define $r(x_{-i}) = r(x_0)$ for $i \geq 1$. From Lemma 4.1, the change in $R(T)$ is at most $3\sum_{i=1}^{k}[r(x_{l(i)-2}) - r(x_{l(i)})] - 2(k-1)$. Since, for all i, we have $l(i+1) \geq l(i)+3$ and $r(x_i) \geq r(x_{i+1})$, we can conclude that

$$\sum_{i=1}^{k}[r(x_{l(i)-2}) - r(x_{l(i)})] \leq r(t) - r(x_{l(k)}).$$

The lemma follows. □

We now prove that the RPS heuristic supports accesses, insertions, and deletions in $O(\log n)$ amortized time.

THEOREM 5.2. *The expected time of m operations using the RPS heuristic with parameter d in an initially empty tree against an oblivious adversary is $O(m + d\log d \cdot I + \sum_{i=1}^{m}\log n_i)$, where n_i is the size of the tree after operation i is performed and I is the total number of insertions. The worst-case time of the m operations is $O(m + d\sum_{i=1}^{m}\log n_i)$.*

Proof. To analyze the expected search time, we use the potential function $\Phi_E = d \cdot R(T)$ and assign a weight of 1 to each node. It is sufficient to show that the expected-amortized time is $O(\log n)$ for an access, $O(\log n + d\log d)$ for an insertion, and $O(\log n)$ for a deletion.

The expected-amortized time to access a node x on level l is the sum of the search time, which is $l+1$, and the change in potential, which Lemma 4.3 implies

is at most $6 \log n - 2l + 2$. Thus, the expected-amortized access time is at most $6 \log n - l + O(1) = O(\log n)$.

The expected-amortized time to insert a node x on level l is the sum of the expected-amortized access time, which is at most $6 \log n - l + O(1)$, and the change in potential caused by the increase in ranks from the insertion, which can be shown to be at most $d[\log l + \log(n/(n-1))]$. It is simple to show that $d \log l < l$ for all $l \geq 4d \log d$, so the expected-amortized insertion time is less than $6 \log n + d \log(4d \log d) + O(1) = O(\log n + d \log d)$.

The expected-amortized time for a deletion is the sum of the expected-amortized access time, which is $O(\log n)$, and the change in potential caused by the decrease in ranks from the deletion, which is negative. Hence, the expected-amortized deletion time is $O(\log n)$.

To analyze the amortized time, we use the potential function $\Phi_A = d/2 \cdot R(T)$. The amortized time of an operation is no greater than the sum of the search time, which is $l + 1$, the increase in potential from the change in ranks in the case of an insertion, and the change in potential from the partial splay. Observe that the potential change after an insertion due to the increase in ranks is at most $d \log(l+1) \leq d \log(n+1)$. Because the RPS heuristic performs at least $\lfloor (l+1)/d \rfloor$ two-promotions, Lemma 5.1 implies that the change in potential from the partial splay is at most $d/2 \cdot [3 \log n - 2\lfloor (l+1)/d \rfloor + 2] = O(d \log n) - l$, so the total amortized time for any single operation is $O(d \log n)$. The theorem follows. □

We finally show that the RPS heuristic is statically competitive against an adaptive (or oblivious) adversary.

THEOREM 5.3. *The expected time of m accesses in a tree of size n using the RPS heuristic with parameter d against an oblivious adversary is $O(m + dn \log n + \sum_{i=1}^{n} [q_i \log(m/q_i)])$, where q_i is the number of times the ith smallest element is accessed; thus, RPS is statically competitive against an oblivious adversary.*

The worst-case time of m accesses in a tree of size n using the RPS heuristic with parameter d against an adaptive adversary is $O(dm + dn \log n + d \sum_{i=1}^{n} [q_i \log(m/q_i)])$; thus, RPS is statically competitive against an adaptive adversary.

Proof. We use the potential function $\Phi = d/2 \cdot R(T)$ to analyze both the expected and amortized times. The theorem follows from Lemmas 4.3 and 5.1 and is similar to the proof of Theorem 4.7. Note that the static competitiveness against an adaptive adversary follows from the amortized-time bound, rather than from the expected-time bound. □

6. Concluding remarks. We investigated the rate of adaptation achievable by a constant-linkage-cost heuristic for maintaining a binary search tree. In particular, we showed that no $o(\log n)$-linkage-cost heuristic can be competitive and we presented three statically competitive, constant-linkage-cost heuristics. We believe that proofs that R2P and LORPS are statically competitive against oblivious adversaries can be modified to establish that they are statically competitive against *adaptive adversaries.*

Many open problems remain. An obvious problem is whether our lower bounds are tight. Another open problem is whether there are matching lower bounds on the competitiveness of promotion-based heuristics and arbitrary heuristics. An area for further work is to investigate lower bounds on the competitiveness of arbitrary heuristics against oblivious adversaries.

Note that none of the first three heuristics attain the (revised) ideal of a statically competitive, worst-case-CLC heuristic that supports updates and uses only constant extra space. Indeed, an open problem is to devise such a heuristic or to prove that none

exists. Another open problem is to devise a statically competitive, worst-case-CLC heuristic that supports updates; it would achieve three of the four requirements of the ideal. Yet another open problem is to devise a statically competitive, amortized-CLC heuristic that supports updates and uses $o(n)$ extra space.

REFERENCES

[1] G. M. Adel'son-Vel'skii and E. M. Landis, *An algorithm for the organization of information*, Soviet Math. Dokl., 3 (1962), pp. 1259–1262.

[2] B. Allen and I. Munro, *Self-organizing binary search trees*, J. Assoc. Comput. Mach., 25 (1978), pp. 526–535.

[3] C. R. Aragon and R. G. Seidel, *Randomized search trees*, in Proc. 30th Annual IEEE Symposium on Foundations of Computer Science, 1989, pp. 540–545.

[4] J. Bell and G. Gupta, *An evaluation of self-adjusting binary search tree techniques*, Software Practice and Experience, 23 (1993), pp. 369–382.

[5] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson, *On the power of randomization in online algorithms*, in Proc. 22th Annual ACM Symposium on Theory of Computing, 1990, pp. 379–386.

[6] S. W. Bent, D. D. Sleator, and R. E. Tarjan, *Biased search trees*, SIAM J. Comput., 14 (1985), pp. 545–568.

[7] J. L. Bentley and C. C. McGeoch, *Amortized analysis of self-organizing sequential search heuristics*, Comm. ACM, 28 (1985), pp. 404–411.

[8] J. R. Bitner, *Heuristics that dynamically organize data structures*, SIAM J. Comput., 8 (1979), pp. 82–110.

[9] R. P. Cheetham, B. J. Oommen, and D. T. H. Ng, *Adaptive structuring of binary search trees using conditional rotations*, Tech. report SCS-TR-126, Carleton University, Ottawa, Canada, October 1987.

[10] R. Cole, *On the dynamic finger conjecture for splay trees (extended abstract)*, in Proc. 22nd Annual ACM Symposium on Theory of Computing, 1990, pp. 8–17.

[11] L. J. Guibas and R. Sedgewick, *A dichromatic framework for balanced trees*, in Proc. 19th Annual IEEE Symposium on Foundations of Computer Science, 1978, pp. 8–21.

[12] T. W. H. Lai, *Efficient Maintenance of Binary Search Trees*, Ph.D. thesis, University of Waterloo, 1990.

[13] T. W. H. Lai and D. Wood, *Adaptive heuristics for binary search trees and constant linkage cost*, in Proc. 2nd Annual ACM-SIAM Symposium on Discrete Algorithms, 1991, pp. 72–77.

[14] E. M. McCreight, *Priority search trees*, SIAM J. Comput., 14 (1985), pp. 257–276.

[15] K. Mehlhorn, *Dynamic binary search*, SIAM J. Comput., 8 (1979), pp. 175–198.

[16] J. Nievergelt and E. M. Reingold, *Binary search trees of bounded balance*, SIAM J. Comput., 2 (1973), pp. 33–43.

[17] T. Ottmann and D. Wood, *Updating binary trees with constant linkage cost*, Internat. J. Found. Comput. Sci., 3 (1992), pp. 479–501.

[18] P. Raghavan and M. Snir, *Memory versus randomization in on-line algorithms*, IBM J. Research and Development, 38 (1994), pp. 683–707.

[19] M. Sherk, *Self-adjusting k-ary Search Trees and Self-adjusting Balanced Search Trees*, Tech. report 234/90, University of Toronto, February 1990.

[20] H.-W. Six and D. Wood, *Counting and reporting intersections of d-ranges*, IEEE Trans. Comput., C-31 (1982), pp. 181–187.

[21] D. D. Sleator and R. E. Tarjan, *Self-adjusting binary search trees*, J. Assoc. Comput. Mach., 32 (1985), pp. 652–686.

[22] R. Wilber, *Lower bounds for accessing binary search trees with rotations*, SIAM J. Comput., 18 (1989), pp. 56–67.

SIAM J. COMPUT.
Vol. 27, No. 6, pp. 1592–1616, December 1998
© 1998 Society for Industrial and Applied Mathematics
005

TREE CONTRACTIONS AND EVOLUTIONARY TREES*

MING-YANG KAO†

Abstract. An *evolutionary tree* is a rooted tree where each internal vertex has at least two children and where the leaves are labeled with distinct symbols representing species. Evolutionary trees are useful for modeling the evolutionary history of species. An *agreement subtree* of two evolutionary trees is an evolutionary tree which is also a topological subtree of the two given trees. We give an algorithm to determine the largest possible number of leaves in any agreement subtree of two trees T_1 and T_2 with n leaves each. If the maximum degree d of these trees is bounded by a constant, the time complexity is $O(n \log^2 n)$ and is within a $\log n$ factor of optimal. For general d, this algorithm runs in $O(nd^2 \log d \log^2 n)$ time or alternatively in $O(nd\sqrt{d} \log^3 n)$ time.

Key words. minimal condensed forms, tree contractions, evolutionary trees, computational biology

AMS subject classifications. 05C05, 05C85, 05C90, 68Q25, 92B05

PII. S0097539795283504

1. Introduction.

An *evolutionary tree* is a rooted tree where each internal vertex has at least two children and where the leaves are labeled with distinct symbols representing species. Evolutionary trees are useful for modeling the evolutionary history of species. Many mathematical biologists and computer scientists have been investigating how to construct and compare evolutionary trees [2, 5, 7, 10, 11, 12, 16, 17, 18, 20, 24, 26, 27, 28, 33, 34, 35, 36, 37, 43, 44, 46, 48, 49]. An *agreement subtree* of two evolutionary trees is an evolutionary tree which is also a topological subtree of the two given trees. A *maximum* agreement subtree is one with the largest possible number of leaves. Different theories about the evolutionary history of the same species often result in different evolutionary trees. A fundamental problem in computational biology is to determine how much two theories have in common. To a certain extent, this problem can be answered by computing a maximum agreement subtree of two given evolutionary trees [19].

Let T_1 and T_2 be two evolutionary trees with n leaves each. Let d be the maximum degree of these trees. Previously, Kubicka, Kubicki, and McMorris [39] gave an algorithm that can compute the number of leaves in a maximum agreement subtree of T_1 and T_2 in $O(n^{(\frac{1}{2}+\epsilon)\log n})$ time for $d = 2$. Steel and Warnow [47] gave the first polynomial-time algorithm. Their algorithm runs in $O(\min\{d!n^2, d^{2.5}n^2 \log n\})$ time if d is bounded by a constant and in $O(n^{4.5} \log n)$ time for general trees. Farach and Thorup [14] later reduced the time complexity of this algorithm to $O(n^2)$ for general trees. More recently, they gave an algorithm [15] that runs in $O(n^{1.5} \log n)$ time for general trees. If d is bounded by a constant, this algorithm runs in $O(nc^{\sqrt{\log n}} + n\sqrt{d} \log n)$ time for some constant $c > 1$.

This paper presents an algorithm for computing a maximum agreement subtree in $O(n \log^2 n)$ time for d bounded by a constant. Since there is a lower bound of $\Omega(n \log n)$, our algorithm is within a $\log n$ factor of optimal. For general d, this algorithm runs in $O(nd^2 \log d \log^2 n)$ time or alternatively in $O(nd\sqrt{d} \log^3 n)$ time.

* Received by the editors March 22, 1995; accepted for publication (in revised form) August 22, 1996; published electronically June 3, 1998.
http://www.siam.org/journals/sicomp/27-6/28350.html

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

This algorithm employs new tree contraction techniques [1, 22, 38, 40, 41]. With tree contraction, we can immediately obtain an $O(n \log^5 n)$-time algorithm for d bounded by a constant. Reducing the time bound to $O(n \log^2 n)$ requires additional techniques. We develop new results that are useful for bounding the time complexity of tree contraction algorithms. As in [14, 15, 47], we also explore the dynamic programming structure of the problem. We obtain some highly regular structural properties and combine these properties with the tree contraction techniques to reduce the time bound by a factor of $\log^2 n$. To remove the last $\log n$ factor, we incorporate some techniques that can compute maxima of multiple sets of sequences at multiple points, where the input sequences are in a compressed format.

We present tree contraction techniques in section 2 and outline our algorithms in section 3. The maximum agreement subtree problem is solved in sections 4 and 5 with a discussion of condensed sequence techniques in section 5.1. Section 6 concludes this paper with an open problem.

2. New tree contraction techniques. Throughout this paper, all trees are rooted ones, and every nonempty tree path is a vertex-simple one from a vertex to a descendant. For a tree T and a vertex u, let T^u denote the subtree of T formed by u and all its descendants in T.

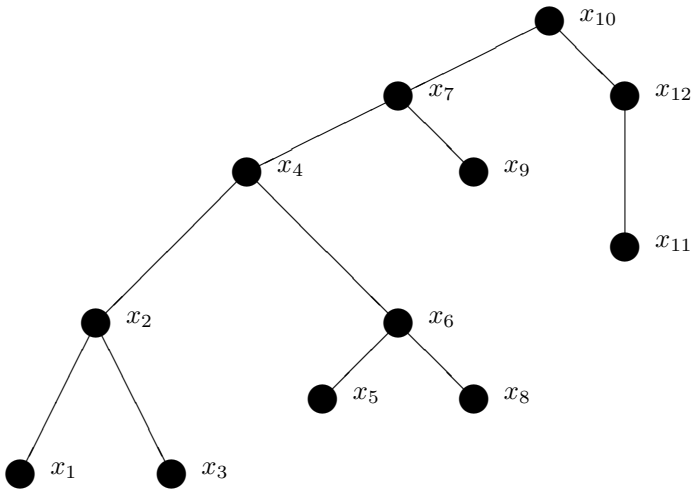
A key idea of our dynamic programming approach is to partition T_1 and T_2 into well-structured tree paths. We recursively solve our problem for T_1^x and T_2^y for all heads x and y of the tree paths in the partitions of T_1 and T_2, respectively. The partitioning is based on new tree contraction techniques developed in this section.

A tree is *homeomorphic* if every internal vertex of that tree has at least two children. Note that the size of a homeomorphic tree is less than twice its number of leaves. Let S be a tree that may or may not be homeomorphic. A *chain of* S is a tree path in S such that every vertex of the given path has at most one child in S. A *tube of* S is a maximal chain of S. A *root path of a tree* is a tree path whose head is the root of that tree; similarly, a *leaf path* is one ending at a leaf. A *leaf tube of* S is a tube that is also a leaf path. Let $\mathcal{L}(S)$ denote the set of leaf tubes in S. Let $\mathcal{R}(S) = S - \mathcal{L}(S)$, i.e., the subtree of S obtained by deleting from S all its leaf tubes. The operation $\mathcal{R}$ is called the *rake operation*. See Figures 1 and 2 for examples of rakes and leaf tubes.

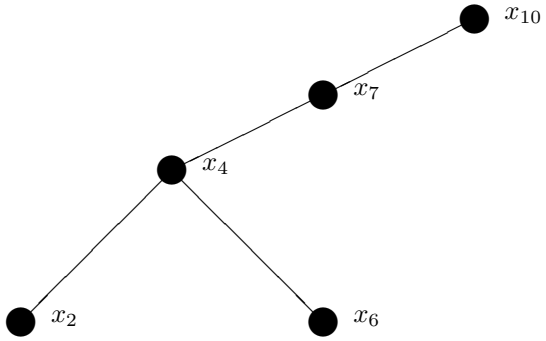
Our dynamic programming approach iteratively rakes T_1 and T_2 until they become empty. The tubes obtained in the process form the desired partitions of T_1 and T_2. Our rake-based algorithms focus on certain sets of tubes described here. A *tube system* of a tree T is a set of nonempty tree paths $P_1, \ldots, P_m$ in T such that (1) the paths P_i contain no leaves of T and (2) $T^{h_1}, \ldots, T^{h_m}$ are pairwise disjoint, where h_i is the head of P_i. Condition (1) is required here because our rake-based algorithms process leaves and nonleaf vertices differently. Condition (2) holds if and only if for all i and j, h_i is not an ancestor or descendant of h_j. We can iteratively rake T to obtain tube systems. The set of tubes obtained by the first rake, i.e., $\mathcal{L}(T)$, is not a tube system of T because $\mathcal{L}(T)$ simply consists of the leaves of T and thus violates condition (1). Every further rake produces a tube system of T until T is raked to empty. Our rake-based algorithms only use these systems although there may be others.

We next develop a theorem to bound the time complexities of rake-based algorithms in this paper. For a tree path P in a tree T,

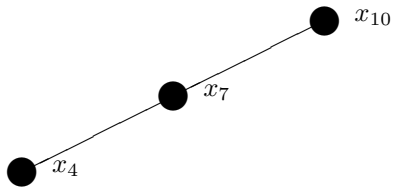
- $K(P, T)$ denotes the set of children of P's vertices in T, excluding P's vertices;
- $t(P)$ denotes the number of vertices in P;
- $b(P, T)$ denotes the number of leaves in T^h where h is the head of P.



After the first rake, the above tree becomes the following tree.

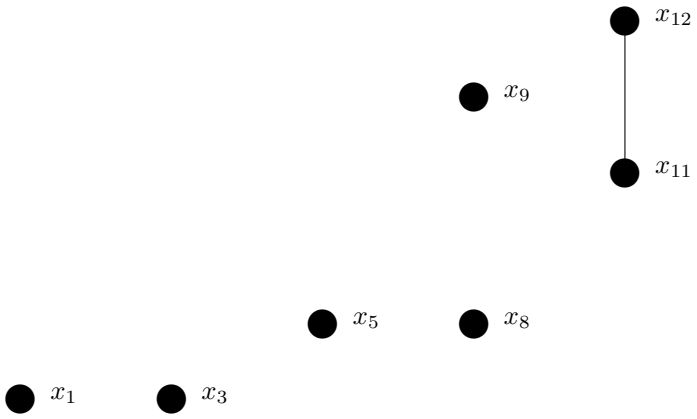


After the second rake, the above tree becomes the following tree.



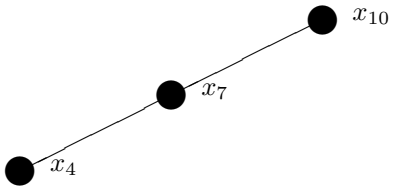
After the third rake, the above tree becomes empty.

FIG. 1. *An example of iterative applications of rakes.*



The first rake deletes the above leaf tubes.

The second rake deletes the above leaf tubes.



The third rake deletes the above leaf tube.

FIG. 2. *The leaf tubes deleted by the rakes in Figure* 1.

(The symbol K stands for the word "kids," t for top, and b for bottom.)

Given T, we recursively define a mapping Φ_T from the subtrees S of T to reals. If S is an empty tree, then $\Phi_T(S) = 0$. Otherwise,

$$\Phi_T(S) = \Phi_T(\mathcal{R}(S)) + \sum_{P \in \mathcal{L}(S)} b(P,T) \cdot \log(1 + t(P)).$$

(*Note.* All logarithmic functions log in this paper are in base 2.)

THEOREM 2.1. *For all positive integers n and all n-leaf homeomorphic trees T, $\Phi_T(T) \le n(1 + \log n)$.*

Proof. For any given n, $\Phi_T(T)$ is maximized when T is a binary tree formed by attaching n leaves to a path of $n-1$ vertices. The proof is by induction.

Base case. For $n = 1$, the theorem trivially holds.

Now assume $n \ge 2$.

Induction hypothesis. For every positive integer $n' < n$, the theorem holds.

Induction step. Let r be the smallest integer such that T is empty after r rakes. Then, at the end of the $(r-1)$th rake, T is a path $P = x_1, \ldots, x_p$. Let $T_1, \ldots, T_s$ be the subtrees of T rooted at vertices in $K(P,T)$. Let n_i be the number of leaves in T_i. Note that

$$\Phi_T(T) = n \log(p+1) + \sum_{i=1}^{s} \Phi_{T_i}(T_i).$$

Since $1 \le n_i < n$ and T_i is homeomorphic, by the induction hypothesis,

$$\Phi_T(T) \le n \log(p+1) + \sum_{i=1}^{s} n_i(1 + \log n_i).$$

Since $\sum_{i=1}^{s} n_i = n$,

$$\Phi_T(T) \le n + n \log(p+1) + \sum_{i=1}^{s} n_i \log n_i. \tag{1}$$

Because T is homeomorphic, each x_i has at least one child in $K(P,T)$. Since $n \ge 2$, $r \ge 2$. Then, x_p cannot be a leaf in T and thus has at least two children in $K(P,T)$. Consequently, $s \ge p+1$. Next, note that for all $m_1, m_2 > 0$,

$$m_1 \log m_1 + m_2 \log m_2 \le (m_1 + m_2) \log(m_1 + m_2).$$

With this inequality and the fact that $s \ge p+1$, we can combine the terms in the right-hand-side summation of inequality (1) to obtain the following inequality:

$$\Phi_T(T) \le n + n \log(p+1) + \sum_{i=1}^{p+1} n'_i \log n'_i, \tag{2}$$

where $\sum_{i=1}^{p+1} n'_i = n$ and $n'_i \ge 1$. For any given p, the summation in inequality (2) is maximized when $n'_1 = n - p$ and $n'_2 = \cdots = n'_{p+1} = 1$. Therefore,

$$\Phi_T(T) \le n + n \log(p+1) + (n-p) \log(n-p). \tag{3}$$

The right-hand side of inequality (3) is maximized when $p = n - 1$. This gives the desired bound and finishes the induction proof. □

3. Comparing evolutionary trees. Formally, an *evolutionary tree* is a homeomorphic tree whose leaves are labeled by distinct labels. The *label set of an evolutionary tree* is the set of all the leaf labels of that tree.

The *homeomorphic version T' of a tree T* is the homeomorphic tree constructed from T as follows. Let $W = \{w \mid w \text{ is a leaf of } T \text{ or is the lowest common ancestor of two leaves}\}$. T' is the tree over W that preserves the ancestor–descendant relationship of T. Let T_1 and T_2 be two evolutionary trees with label sets L_1 and L_2, respectively.

- For a subset L_1' of L_1, $T_1||L_1'$ denotes the homeomorphic version of the tree constructed by deleting from T_1 all the leaves with labels outside L_1'.
- Let $T_1||T_2 = T_1||(L_1 \cap L_2)$.
- For a tree path P of T_1, $P||T_2$ denotes the tree path in $T_1||T_2$ formed by the vertices of P that remain in $T_1||T_2$.
- For a set $\mathcal{P}$ of tree paths $P_1, \ldots, P_m$ of T_1, $\mathcal{P}||T_2$ denotes the set of all $P_i||T_2$.

Formally, if L' is a maximum cardinality subset of $L_1 \cap L_2$ such that there exists a label-preserving tree isomorphism between $T_1||L'$ and $T_2||L'$, then $T_1||L'$ and $T_2||L'$ are called *maximum agreement subtrees of T_1 and T_2*.

- $\text{RR}(T_1, T_2)$ denotes the number of leaves in a maximum agreement subtree of T_1 and T_2.
- $\text{RA}(T_1, T_2)$ is the mapping from each vertex $v \in T_2||T_1$ to $\text{RR}(T_1, (T_2||T_1)^v)$, i.e., $\text{RA}(T_1, T_2)(v) = \text{RR}(T_1, (T_2||T_1)^v)$.

For a tree path Q of T_2, if Q is nonempty, let $H(Q, T_2)$ be the set of all vertices in Q and those in $K(Q, T_2)$. If Q is empty, let $H(Q, T_2)$ consist of the root of T_2 and, thus, if both T_2 and Q are empty, $H(Q, T_2) = \emptyset$.

- For a set $\mathcal{Q}$ of tree paths $Q_1, \ldots, Q_m$ of T_2, let $\text{RP}(T_1, T_2, \mathcal{Q})$ be the mapping from $v \in \cup_{i=1}^{m} H(Q_i||T_1, T_2||T_1)$ to $\text{RR}(T_1, (T_2||T_1)^v)$, i.e., $\text{RP}(T_1, T_2, \mathcal{Q})(v) = \text{RR}(T_1, (T_2||T_1)^v)$. For simplicity, when $\mathcal{Q}$ consists of only one path Q, let $\text{RP}(T_1, T_2, Q)$ denote $\text{RP}(T_1, T_2, \mathcal{Q})$.

(The notations RR, RA, and RP stand for the phrases root to root, root to all, and root to path. We use RR to replace the notation MAST of previous work [14, 15, 47] for the sake of notational uniformity.)

LEMMA 3.1. *Let T_1, T_2, T_3 be evolutionary trees.*

- $(T_1||T_2)||T_3 = T_1||(T_2||T_3)$.
- *If T_3 is a subtree of T_1, then $T_3||T_1 = T_1||T_3 = T_3$.*
- $\text{RR}(T_1, T_2) = \text{RR}(T_1||T_2, T_2) = \text{RR}(T_1, T_2||T_1) = \text{RR}(T_1||T_2, T_2||T_1)$.

Proof. The proof is straightforward. □

FACT 1 (see [14]). *Given an n-leaf evolutionary tree T and k disjoint sets $L_1, \ldots, L_k$ of leaf labels of T, the subtrees $T||L_1, \ldots, T||L_k$ can be computed in $O(n)$ time.*

Proof. The ideas are to preprocess T for answering queries of lowest common ancestors [25, 45] and to reconstruct subtrees from appropriate tree traversal numbering [4, 9]. □

Given T_1 and T_2, our main goal is to evaluate $\text{RR}(T_1, T_2)$ efficiently. Note that $\text{RR}(T_1, T_2) = \text{RR}(T_1||T_2, T_2||T_1)$ and that $T_1||T_2$ and $T_2||T_1$ can be computed in linear time. Thus, the remaining discussion assumes that T_1 and T_2 have the same label set. To evaluate $\text{RR}(T_1, T_2)$, we actually compute $\text{RA}(T_2, T_1)$ and divide the discussion among the five problems defined below. Each problem is named as a p-q case, where p and q are the numbers of tree paths in T_1 and T_2 contained in the input. The inputs of these problems are illustrated in Figure 3.

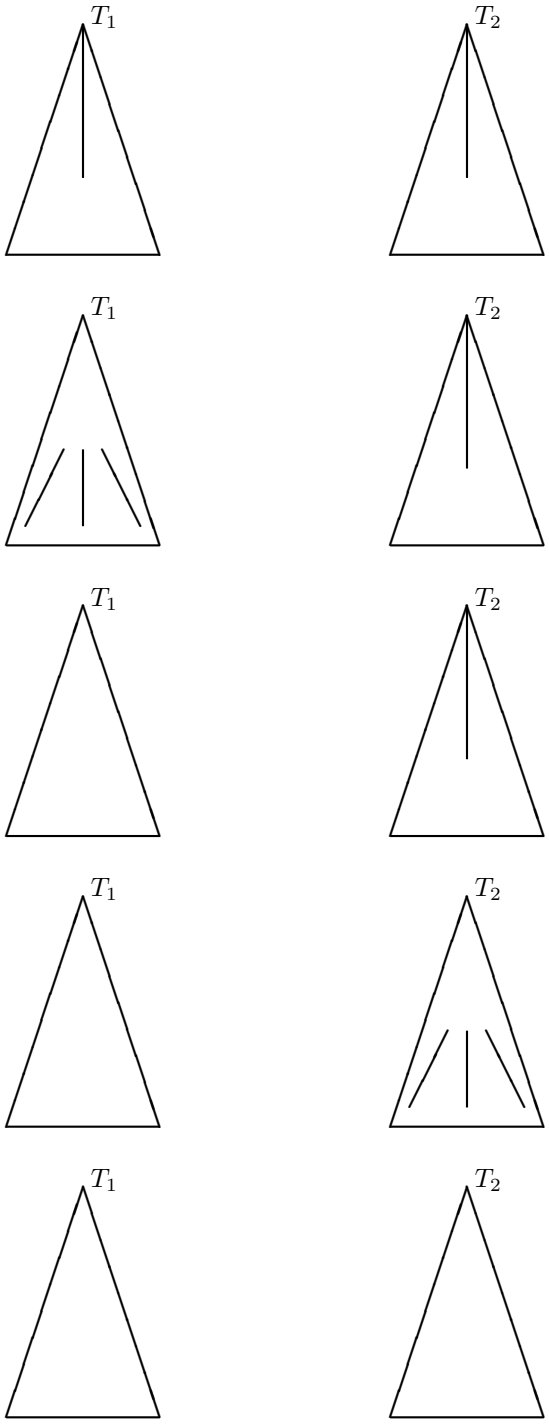


FIG. 3. *Inputs of Problems* 1–5.

PROBLEM 1 (ONE-ONE CASE).

Input:

1. T_1 and T_2;
2. root paths P of T_1 and Q of T_2 with no leaves from their respective trees;
3. $\text{RP}(T_1^u, T_2, Q)$ for all $u \in K(P, T_1)$;
4. $\text{RP}(T_2^v, T_1, P)$ for all $v \in K(Q, T_2)$.

Output: $\text{RP}(T_1, T_2, Q)$ and $\text{RP}(T_2, T_1, P)$.

The next problem generalizes Problem 1.

PROBLEM 2 (MANY-ONE CASE).

Input:

1. T_1 and T_2;
2. a tube system $\mathcal{P} = \{P_1, \ldots, P_m\}$ of T_1 and a root path Q of T_2 with no leaf from T_2;
3. $\text{RP}(T_1^u, T_2, Q)$ for all P_i and $u \in K(P_i, T_1)$;
4. $\text{RP}(T_2^v, T_1, \mathcal{P})$ for all $v \in K(Q, T_2)$.

Output:

1. $\text{RP}(T_1^{h_i}, T_2, Q)$ for the head h_i of each P_i;
2. $\text{RP}(T_2, T_1, \mathcal{P})$.

PROBLEM 3 (ZERO-ONE CASE).

Input:

1. T_1 and T_2;
2. a root path Q of T_2 with no leaf from T_2;
3. $\text{RA}(T_2^v, T_1)$ for all $v \in K(Q, T_2)$.

Output: $\text{RA}(T_2, T_1)$.

The next problem generalizes Problem 3.

PROBLEM 4 (ZERO-MANY CASE).

Input:

1. T_1 and T_2;
2. a tube system $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$ of T_2;
3. $\text{RA}(T_2^v, T_1)$ for all Q_i and $v \in K(Q_i, T_2)$.

Output: $\text{RA}(T_2^{h_i}, T_1)$ for the head h_i of each Q_i.

Our main goal is to evaluate $\text{RR}(T_1, T_2)$. It suffices to solve the next problem.

PROBLEM 5 (ZERO-ZERO CASE).

Input: T_1 and T_2.

Output: $\text{RA}(T_2, T_1)$.

Our algorithms for these problems are called *One-One, Many-One, Zero-One, Zero-Many*, and *Zero-Zero*, respectively. Each algorithm except One-One uses the preceding one in this list as a subroutine. These reductions are based on the rake operation defined in section 2. We give One-One in section 5 and the other four in sections 4.1–4.4.

These five algorithms assume that the input trees T_1 and T_2 have n leaves each and d is the maximum degree. We use integer sort and radix sort [4, 9] extensively to help achieve the desired time complexity. (For brevity, from here onward, radix sort refers to both integer and radix sorts.) For this reason, we make the following *integer indexing assumptions*:

- an integer array of size $O(n)$ is allocated to each algorithm;
- the vertices of T_1 and T_2 are indexed by integers from $[1, O(n)]$;
- the leaf labels are indexed by integers from $[1, O(n)]$.

We call Zero-Zero only once to compare two given trees. Consequently, we may

reasonably assume that the tree vertices are indexed with integers from $[1, O(n)]$. When we call Zero-Zero, we simply allocate an array of size $O(n)$. As for indexing the leaf labels, this paper considers only evolutionary trees whose leaf labels are drawn from a total order. Before we call Zero-Zero, we can sort the leaf labels and index them with integers from $[1, O(n)]$. This preprocessing takes $O(n \log n)$ time, which is well within our desired time complexity for Zero-Zero.

The other four algorithms are called more than once, and their integer indexing assumptions are maintained in slightly different situations from that for Zero-Zero. When an algorithm issues subroutine calls, it is responsible for maintaining the indexing assumptions for the callees. In certain cases, the caller uses radix sort to reindex the labels and the vertices of each callee's input trees. The caller also partitions its array into segments and allocates to each callee a segment in proportion to that callee's input size. The new indices and the array segments for subroutine calls can be computed in obvious manners within the desired time complexity of each caller. For brevity of presentation, such preprocessing steps are omitted in the descriptions of the five algorithms.

Some inputs to the algorithms are mappings. We represent a mapping f by the set of all pairs $(x, f(x))$. With this representation, the total size of the input mappings in an algorithm is $O(n)$. Since the input mappings have integer values at most n, this representation and the integer indexing assumptions together enable us to evaluate the input mappings at many points in a batch by means of radix sort. Other mappings that are produced within the algorithms are similarly evaluated. When these algorithms are detailed, it becomes evident that such evaluations can computed in straightforward manners in time linear in n and the number of points evaluated. The descriptions of these algorithms assume that the values of mappings are accessed by radix sort.

4. The rake-based reductions. For ease of understanding, our solutions to Problems 1–5 are presented in a different order from their logical one. This section assumes the following theorem for Problem 1 and uses it to solve Problems 2–5. In section 5.6, we prove this theorem by giving an algorithm, called *One-One*, that solves Problem 1 within the theorem's stated time bounds.

THEOREM 4.1. *Problem* 1 *can be solved in* $O(nd^2 \log d + n \log(p+1) \log(q+1))$ *time or alternatively in* $O(nd\sqrt{d} \log n + n \log(p+1) \log(q+1))$ *time.*

Proof. The proof follows from theorem 5.14 at the end of section 5.6. □

4.1. The many-one case. The following algorithm is for Problem 2 and uses One-One as a subroutine. Note that Problem 2 is merely a multipath version of Problem 1.

ALGORITHM MANY-ONE

begin

1. For all P_i, compute $T_{1,i} = T_1^{h_i}$, $T_{2,i} = T_2 || T_{1,i}$, and $Q_i = Q || T_{1,i}$;
2. For all empty Q_i, compute part of the output as follows:
   (a) Compute the root $\hat{v}$ of $T_{2,i}$ and $v \in K(Q, T_2)$ such that $\hat{v} \in T_2^v$;
   (b) $\text{RP}(T_1^{h_i}, T_2, Q)(\hat{v}) \leftarrow \text{RP}(T_2^v, T_1, \mathcal{P})(h_i)$; (*Note.* $H(Q_i, T_{2,i}) = \{\hat{v}\}$. This is part of the output.)
   (c) For all $x \in H(P_i, T_1)$, $\text{RP}(T_2, T_1, \mathcal{P})(x) \leftarrow \text{RP}(T_2^v, T_1, \mathcal{P})(x)$; (*Note.* This is part of the output.)
3. For all nonempty Q_i, compute the remaining output as follows: (*Note.* The many-one case is reduced to the one-one case with input $T_{1,i}$, $T_{2,i}$, P_i, and Q_i.)

(a) For all $u \in K(P_i, T_{1,i})$, $\text{RP}(T^u_{1,i}, T_{2,i}, Q_i) \leftarrow \text{RP}(T^u_1, T_2, Q)$;
(b) For all $\hat{v} \in K(Q_i, T_{2,i})$, compute $\text{RP}(T^{\hat{v}}_{2,i}, T_{1,i}, P_i)$ as follows:
  i. Compute the vertex $v \in K(Q, T_2)$ such that $\hat{v} \in T^v_2$;
  ii. $\text{RP}(T^{\hat{v}}_{2,i}, T_{1,i}, P_i)(x) \leftarrow \text{RP}(T^v_2, T_1, \mathcal{P})(x)$ for all $x \in H(P_i, T_{1,i})$;
(c) Compute $\text{RP}(T_{1,i}, T_{2,i}, Q_i)$ and $\text{RP}(T_{2,i}, T_{1,i}, P_i)$ by applying One-One to $T_{1,i}, T_{2,i}$, P_i, Q_i, and the mappings computed at steps 3(a) and 3(b);
(d) $\text{RP}(T^{h_i}_1, T_2, Q) \leftarrow \text{RP}(T_{1,i}, T_{2,i}, Q_i)$; (*Note.* This is part of the output.)
(e) For all $x \in H(P_i, T_{1,i})$, $\text{RP}(T_2, T_1, \mathcal{P})(x) \leftarrow \text{RP}(T_{2,i}, T_{1,i}, P_i)(x)$; (*Note.* This is part of the output.)

end.

THEOREM 4.2. *Algorithm Many-One solves Problem* 2 *with the following time complexities:*

$$O\left(nd^2 \log d + \log(1 + t(Q)) \cdot \sum_{i=1}^{m} b(P_i, T_1) \log(1 + t(P_i))\right)$$

or, alternatively,

$$O\left(nd\sqrt{d} \log n + \log(1 + t(Q)) \cdot \sum_{i=1}^{m} b(P_i, T_1) \log(1 + t(P_i))\right).$$

Proof. Since T_1 and T_2 have the same label set, all $T_{2,i}$ are nonempty. To compute the output RP, there are two cases depending on whether Q_i is empty or nonempty. These cases are computed by steps 2 and 3. The correctness of Many-One is then determined by that of steps 2(b), 2(c), 3(a), 3(b), 3(b)ii, 3(d), and 3(e). These steps can be verified using Lemma 3.1. As for the time complexity, these steps take $O(n)$ time using radix sort to evaluate RP. Step 1 uses Fact 1 and takes $O(n)$ time. Steps 2(a) and 3(b)i take $O(n)$ time using tree traversal and radix sort. As discussed in section 3, step 3(c) preprocesses the input of its One-One calls to maintain their integer indexing assumptions. We reindex the labels and vertices of $T_{1,i}$ and $T_{2,i}$ and pass the new indices to the calls. We also partition Many-One's $O(n)$-size array to allocate a segment of size $|T_{1,i}|$ to the call with input $T_{1,i}$. Since the total input size of the calls is $O(n)$, this preprocessing takes $O(n)$ time in an obvious manner. After this preprocessing, the running time of step 3(c) dominates that of Many-One. The stated time bounds follow from Theorem 4.1 and the fact that Q_i is not longer than Q and the degrees of $T_{2,i}$ are at most d. □

4.2. The zero-one case. The following algorithm is for Problem 3. It uses Many-One as a subroutine to recursively compare T_2 with the subtrees of T_1 rooted at the heads of the tubes obtained by iteratively raking T_1. The tubes obtained by the first rake are compared with T_2 first, and the tube obtained by the last rake is compared last.

ALGORITHM ZERO-ONE

begin

1. $S \leftarrow T_1$;
2. $LF \leftarrow \mathcal{L}(S)$; (*Note.* LF consists of the leaves of T_1.)
3. For all $x \in LF$, $\text{RA}(T_2, T_1)(x) \leftarrow 1$; (*Note.* This is part of the output.)
4. For all $u \in LF$, $\text{RP}(T^u_1, T_2, Q)(y) \leftarrow 1$, where y is the unique vertex of $T_2 || T^u_1$; (*Note.* This is the base case of rake-based recursion.)
5. $S \leftarrow S - \mathcal{L}(S)$;

6. **while** S is not empty **do** the following steps:
   (a) Compute $\mathcal{L}(S) = \{P_1, \ldots, P_m\}$;
   (b) Gather the mappings $\text{RP}(T_1^u, T_2, Q)$ for all P_i and $u \in K(P_i, T_1)$; (*Note.* These mappings are either initialized at step 4 or computed at previous iterations of step 6(d).)
   (c) $\text{RP}(T_2^v, T_1, \mathcal{L}(S))(x) \leftarrow \text{RA}(T_2^v, T_1)(x)$ for all $v \in K(Q, T_2)$ and $x \in \cup_{i=1}^m H(P_i, T_1)$;
   (d) Compute $\text{RP}(T_1^{h_i}, T_2, Q)$ for the head h_i of each P_i and $\text{RP}(T_2, T_1, \mathcal{L}(S))$ by applying Many-One to T_1, T_2, $\mathcal{L}(S)$, Q, and the mappings obtained at steps 6(b) and 6(c); (*Note.* This is the recursion step of rake-based recursion.)
   (e) For all $x \in \cup_{i=1}^m K(P_i, T_1)$, $\text{RA}(T_2, T_1)(x) \leftarrow \text{RP}(T_2, T_1, \mathcal{L}(S))(x)$; (*Note.* This is part of the output.)
   (f) $S \leftarrow S - \mathcal{L}(S)$;

end.

THEOREM 4.3. *Zero-One solves Problem* 3 *with the following time complexities:*

$$O(nd^2 \log d \log n + n \log n \log(1 + t(Q)))$$

or, alternatively,

$$O(nd\sqrt{d} \log^2 n + n \log n \log(1 + t(Q))).$$

Proof. The $\mathcal{L}(S)$ at step 6(a) is a tube system. The heads of the tubes in $\mathcal{L}(S)$ become children of the tubes in future $\mathcal{L}(S)$. The vertices $u \in K(P_i, T_1)$ at step 6(b) are either leaves of T_1 or heads of the tubes in previous $\mathcal{L}(S)$. These properties ensure the correctness of the rake-based recursion. The remaining correctness proof uses Lemma 3.1 to verify the correctness of steps 3, 4, 6(c) and 6(e). Steps 1–5, 6(a), 6(b), and 6(f) are straightforward and take $O(n)$ time. Steps 6(c) and 6(e) take $O(n)$ time using radix sort to access RP and RA. At Step 6(d), to maintain the integer indexing assumptions for the call to Many-One, we simply pass to Many-One the indices of T_1 and T_2 and the whole array of Zero-One. Step 6(d) has the same time complexity as Zero-One. The desired time bounds follow from Theorems 2.1 and 4.2. □

4.3. The zero-many case. The following algorithm is for Problem 4 and uses Zero-One as a subroutine. Note that Problem 4 is merely a multipath version of Problem 3.

ALGORITHM ZERO-MANY

begin

1. For all Q_i, compute $T_{2,i} = T_2^{h_i}$ and $T_{1,i} = T_1 || T_{2,i}$;
2. For all Q_i and $v \in K(Q_i, T_{2,i})$, $\text{RA}(T_{2,i}^v, T_{1,i}) \leftarrow \text{RA}(T_2^v, T_1)$;
3. For all Q_i, compute $\text{RA}(T_{2,i}, T_{1,i})$ by applying Zero-One to $T_{1,i}, T_{2,i}$, Q_i, and the mapping computed at Step 2;
4. For all $Q_i, \text{RA}(T_2^{h_i}, T_1) \leftarrow \text{RA}(T_{2,i}, T_{1,i})$; (*Note.* This is the output.)

end.

THEOREM 4.4. *Zero-Many solves Problem* 4 *with the following time complexities:*

$$O\left(nd^2 \log d \log n + \log n \cdot \sum_{i=1}^{m} b(Q_i, T_2) \log(1 + t(Q_i))\right)$$

or, alternatively,

$$O\left(nd\sqrt{d}\log^2 n + \log n \cdot \sum_{i=1}^{m} b(Q_i, T_2)\log(1 + t(Q_i))\right).$$

Proof. The proof is similar to that of Theorem 4.2. The time bounds follow from Theorem 4.3. □

4.4. The zero-zero case. The following algorithm is for Problem 5. It uses Zero-Many as a subroutine to recursively compare T_1 with the subtrees of T_2 rooted at the heads of the tubes obtained by iteratively raking T_2. The tubes obtained by the first rake are compared with T_1 first, and the tube obtained by the last rake is compared last.

ALGORITHM ZERO-ZERO

begin

1. $S \leftarrow T_2$;
2. $LF \leftarrow \mathcal{L}(S)$; (*Note.* LF consists of the leaves of T_2.)
3. For all $v \in LF$, $\text{RA}(T_2^v, T_1)(x) \leftarrow 1$, where x is the only vertex in $T_1 || T_2^v$; (*Note.* This is the base case of rake-based recursion.)
4. $S \leftarrow S - \mathcal{L}(S)$;
5. **while** S is not empty **do**
   (a) Compute $\mathcal{L}(S) = \{Q_1, \ldots, Q_m\}$;
   (b) Gather the mappings $\text{RA}(T_2^v, T_1)$ for all Q_i and $v \in K(Q_i, T_2)$; (*Note.* These mappings are either initialized at step 3 or computed at previous iterations of step 5(c).)
   (c) Compute $\text{RA}(T_2^{h_i}, T_1)$ for the head h_i of each Q_i by applying Zero-Many to $T_1, T_2, \mathcal{L}(S)$ and the mappings obtained at step 5(b). (*Note.* This is the recursion step of rake-based recursion.)
   (d) $S \leftarrow S - \mathcal{L}(S)$;
6. $\text{RA}(T_2, T_1) \leftarrow \text{RA}(T_2^h, T_1)$, where h is the root of T_2; (*Note.* This is the output. If T_2 has only one vertex, $\text{RA}(T_2^h, T_1)$ is computed at step 3; otherwise it is computed at the last iteration of step 5(c).)

end.

THEOREM 4.5. *Zero-Zero solves Problem* 5 *within* $O(nd^2 \log d \log^2 n)$ *time or alternatively within* $O(nd\sqrt{d}\log^3 n)$ *time.*

Proof. The proof is similar to that of Theorem 4.3. The time bounds follow from Theorems 2.1 and 4.4. □

5. The one-one case. Our algorithm for Problem 1 makes extensive use of bisection-based dynamic programming and implicit computation in compressed formats. This problem generalizes the longest common subsequence problem [6, 23, 29, 30, 32], which has efficient dynamic programming solutions. A direct dynamic programming approach to our problem would recursively solve the problem with T_1^x and T_2^y in place of T_1 and T_2 for all vertices $x \in P$ and $y \in Q$. This approach may require solving $\Omega(n^2)$ subproblems. To improve the time complexity, observe that the number of leaves in a maximum agreement subtree of T_1^x and T_2^y can range only from 0 to n. Moreover, this number never increases when x moves from the root of T_1 along P to P's endpoint, and y remains fixed, or vice versa. Compared with the length of P, $\text{RR}(T_1^x, T_2^y)$ often assumes relatively few different values. Thus, to compute this number along P, it is useful to compute the locations at P where the number decreases.

We can find those locations with a bisection scheme and use them to implicitly solve the $O(n^2)$ subproblems in certain compressed formats. We first describe basic techniques used in such implicit computation in section 5.1 and then proceed to discuss bisection-based dynamic programming techniques in sections 5.2–5.5. We combine all these techniques to give an algorithm to solve Problem 1 in section 5.6.

5.1. Condensed sequences. For integers k_1 and k_2 with $k_1 \le k_2$, let $[k_1, k_2] = \{k_1, \ldots, k_2\}$, i.e., the integer interval between k_1 and k_2. The *length* of an integer interval is the number of its integers. The *upper* and *lower halves* of an even length $[k_1, k_2]$ are $[k_1, \frac{k_1+k_2-1}{2}]$ and $[\frac{k_1+k_2+1}{2}, k_2]$, respectively. The *regular* integer intervals are defined recursively. For all integers $\alpha \ge 0$, $[1, 2^\alpha]$ is regular. The upper and lower halves of an even length regular interval are also regular.

For example, $[1, 8]$ is regular. Its regular subintervals are $[1, 4]$, $[5, 8]$, $[1, 2]$, $[3, 4]$, $[5, 6]$, $[7, 8]$, and the singletons $[1, 1], [2, 2], \ldots, [8, 8]$.

A *normal sequence* is a nonincreasing sequence $\{f(j)\}_{j=1}^{l}$ of nonnegative numbers. A normal sequence is *nontrivial* if it has at least one nonzero term.

For example, $5, 4, 4, 0$ is a nontrivial normal sequence, whereas $0, 0, 0$ is a trivial one.

Let $f_1, \ldots, f_k$ be k normal sequences of length l. An *interval query* for $f_1, \ldots, f_k$ is a pair $([k_1, k_2], j)$ where $[k_1, k_2] \subseteq [1, k]$ and $j \in [1, l]$. If $k_1 = k_2$, $([k_1, k_2], j)$ is also called a *point query*. The *value* of a query $([k_1, k_2], j)$ is $\max_{k_1 \le i \le k_2} f_i(j)$. A query $([k_1, k_2], j)$ is *regular* if $[k_1, k_2]$ is a regular integer interval.

For example, let

$$\begin{aligned} f_1 &= 5, 4, 4, 3, 2; \\ f_2 &= 8, 7, 4, 2, 0; \\ f_3 &= 9, 9, 5, 0, 0. \end{aligned}$$

Then, f_1, f_2, and f_3 are normal sequences of length 5. Here, $k = 3$ and $l = 5$. Thus, $([1, 3], 2)$ is an interval query; its value is $\max\{f_1(2), f_2(2), f_3(2)\} = 9$. The pair $([1, 1], 3)$ is a point query; its value is $f_1(3) = 4$. The pair $([1, 2], 2)$ is a regular query; its values is $\max\{f_1(2), f_2(2)\} = 7$.

The *joint* of $f_1, \ldots, f_k$ is the normal sequence $\hat{f}$ also of length l such that $\hat{f}(j) = \max\{f_1(j), \ldots, f_k(j)\}$.

Continuing the above example, the joint of f_1, f_2, f_3 is

$$\hat{f} = 9, 9, 5, 3, 2.$$

The *minimal condensed form* of a normal sequence $\{f(j)\}_{j=1}^{l}$ is the set of all pairs $(j, f(j))$ where $f(j) \ne 0$ and j is the largest index of any $f(j')$ with $f(j') = f(j)$. A *condensed form* is a set of pairs $(j, f(j))$ that includes the minimal condensed form. The *size* of a condensed form is the number of pairs in it. The *total size* of a collection of condensed forms is the sum of the sizes of those forms.

Continuing the above example, the minimal condensed form of f_3 is $\{(2, 9), (3, 5)\}$; its size is 2. The set $\{(1, 9), (2, 9), (3, 5), (5, 0)\}$ is a condensed form of f_3; its size is 4. The total size of these two forms is 6.

LEMMA 5.1. *Let $F_1, \ldots, F_k$ be sets of nontrivial normal sequences of length l. Let $\hat{f}_i$ be the joint of the sequences in F_i. Given a condensed form of each sequence in each F_i, we can compute the minimal condensed forms of all $\hat{f}_i$ in $O(l + s)$ time where s is the total size of the input forms.*

Proof. The desired minimal forms can be computed by the two steps below:

1. Sort the pairs in the given condensed forms for F_i into a sequence in the increasing order of the first components of these pairs.
2. Go through this sequence to delete all unnecessary pairs to obtain the minimal condensed form of $\hat{f}_i$.

We can use radix sort to implement step 1 in $O(l+s)$ time for all F_i. Step 2 can be easily implemented in $O(s)$ time for all F_i. □

LEMMA 5.2. *Let $f_1, \ldots, f_k$ be nontrivial normal sequences of length l. Assume that the input consists of a condensed form of each f_i with a total size of s.*

1. *We can evaluate m point queries in $O(m+l+s)$ time.*
2. *We can evaluate m_1 regular queries and m_2 irregular queries in a total of $O(m_1 + (m_2 + l + s)\log(k+1))$ time.*

Proof. The proof of statement 1 uses radix sort in an obvious manner. To prove statement 2, we assume without loss of generality that k is a power of two. The input queries can be evaluated by the following three stages within the desired time bound.

Stage 1. For each regular interval $[k_1, k_2] \subseteq [1, k]$, let $f[k_1, k_2]$ be the joint of $f_{k_1}, \ldots, f_{k_2}$. We use Lemma 5.1 $O(\log(k+1))$ times to compute the minimal condensed forms of all $f[k_1, k_2]$. The total size of these forms is $O(s\log(k+1))$. This stage takes $O((l+s)\log(k+1))$ time.

Stage 2. For each irregular input query $([i_1, i_2], j)$, we partition $[i_1, i_2]$ into $O(\log(k+1))$ regular subintervals $[h_1, h_2], [h_2+1, h_3], \ldots, [h_{r-1}+1, h_r]$. Then, the value of $([i_1, i_2], j)$ is the maximum of those of $([h_1, h_2], j), \ldots, ([h_{r-1}+1, h_r], j)$. These regular queries are point queries for $f[h_1, h_2], \ldots, f[h_{r-1}+1, h_r]$. Together with the given m_1 regular queries, we have now generated $O(m_1 + m_2\log(k+1))$ point queries for all $f[k_1.k_2]$. This stage takes $O(m_1 + m_2\log(k+1))$ time.

Stage 3. We use statement 1 and the minimal condensed forms of $f[k_1.k_2]$ to evaluate the points queries generated at Stage 2. Once the values of these point queries are obtained, we can easily compute the values of the input queries. This stage takes $O(m_1 + m_2\log(k+1) + l + s\log(k+1))$ time. □

5.2. Normalizing the input. To solve Problem 1, we first augment its input T_1, T_2, P, and Q in order to simplify our discussion. Let $P = x_1, \ldots, x_p$ and $Q = y_1, \ldots, y_q$. Without loss of generality, we assume that $p \geq q$.

1. Let α and β be the smallest positive integers such that $p' = 2^\alpha + 1$, $q' = 2^\beta + 1$, $p' \geq q'$, $p' > p$, and $q' > q$. (*Note.* The conditions $p' > p$ and $q' > q$ are employed for technical simplicity. They can be changed to $p' \geq p$ and $q' \geq q$ with some modification to Algorithm One-One.)
2. Attach to x_p the path $x_{p+1}, \ldots, x_{p'}$ and to y_q the path $y_{q+1}, \ldots, y_{q'}$.
3. Let $P' = x_1, \ldots, x_{p'}$ and $Q' = y_1, \ldots, y_{q'}$.
4. Attach a leaf to each of $x_{p+1}, \ldots, x_{p'-1}$ and $y_{q+1}, \ldots, y_{q'-1}$, two leaves to $x_{p'}$, and two leaves to $y_{q'}$.
5. Assign distinct labels to the new leaves which also differ from the existing labels of T_1 and T_2.
6. Let S_1 be T_1 together with P' and the new leaves of P'. Let S_2 be T_2 together with Q' and the new leaves of Q'.

S_1 and S_2 are evolutionary trees. P' and Q' contain no leaves from S_1 and S_2 and are root paths of these trees. Let $n' = \max\{n_1, n_2\}$ where n_i is the number of leaves in S_i. Let d' be the maximum degree in S_1 and S_2.

LEMMA 5.3.

- $n' = O(n)$, $p' = O(p)$, $q' = O(q)$, *and* $d' \leq d+1$.
- $\text{RP}(T_1, T_2, Q) = \text{RP}(S_1, S_2, Q')$ *and* $\text{RP}(T_2, T_1, P) = \text{RP}(S_2, S_1, P')$.

Proof. The proof is straightforward. □

In light of Lemma 5.3, our discussion below mainly works with S_1, S_2, P', and Q'. Let $G = G_P \cup G_Q$ where G_P is the set of all pairs (x_i, y_1) and G_Q is the set of all (x_1, y_j). To solve Problem 1, a main task is to evaluate $\text{RR}(S_1^x, S_2^y)$ for $(x, y) \in G$. The output RP values that are excluded here can be retrieved directly from the input RP mappings.

5.3. Predecessors. A pair $(x_{i'}, y_{j'})$ is a *predecessor* of a distinct (x_i, y_j) if $i \le i'$ and $j \le j'$. One-One proceeds by recursively reducing the problem of computing $\text{RR}(S_1^x, S_2^y)$ to that of computing the RR values of the *P-predecessor, Q-predecessor, and PQ-predecessor* defined below.

Let $P[i, i']$ be the path $x_i, \ldots, x_{i'}$, where $i \le i'$. Let X_i be the set of the children of x_i in S_1 that are not in P'. We similarly define $Q[j, j']$ and Y_j. A pair (x_i, y_j) is *intersecting* if S_1^u and S_2^v have at least one common leaf label for some $u \in X_i$ and $v \in Y_j$. $(P[i, i'], Q[y_j, y_{j'}])$ is *intersecting* if some $x_{i''} \in P[i, i']$ and $y_{i''} \in Q[j, j']$ form an intersecting pair.

The *lengths* of $P[i, i']$ and $Q[j, j']$ are those of $[i, i']$ and $[j, j']$, respectively. A path $P[i, i']$ is *regular* if $[i, i']$ is a regular interval. A *regular* $Q[j, j']$ is similarly defined. We now construct a tree Ψ over pairs of regular paths; this tree is slightly different from that of [15]. The root of Ψ is $(P[1, p' - 1], Q[1, q' - 1])$. A pair $(P[i, i'], Q[j, j']) \in \Psi$ is a leaf if and only if either (1) $i = i'$, $j = j'$, and (x_i, y_j) is intersecting, or (2) this pair is nonintersecting. For a nonleaf $(P[i, i'], Q[j, j']) \in \Psi$, if $j = j'$, then its children are $(P[i, \frac{i+i'-1}{2}], y_j)$ and $(P[\frac{i+i'+1}{2}, i'], y_j)$. Otherwise, this pair has four children $(P[i, \frac{i+i'-1}{2}], Q[j, \frac{j+j'-1}{2}])$, $(P[i, \frac{i+i'-1}{2}], Q[\frac{j+j'+1}{2}, j'])$, $(P[\frac{i+i'+1}{2}, i'], Q[j, \frac{j+j'-1}{2}])$, $(P[\frac{i+i'+1}{2}, i'], Q[\frac{j+j'+1}{2}, j'])$.

The *ceiling of* $(P[i, i'], Q[j, j'])$ is (x_i, y_j); its *floor* is $(x_{i'+1}, y_{j'+1})$ [15]. Its *P-diagonal* is $(x_{i'+1}, y_j)$; its *Q-diagonal* is $(x_i, y_{j'+1})$. Let E be the set of all ceilings, diagonals, floors of the leaves of Ψ. Let $B = \{(x_i, y_{q'}) \mid i \in [1, p']\} \cup \{(x_{p'}, y_j) \mid j \in [1, q']\}$. Due to its recursive nature, One-One evaluates $\text{RR}(S_1^x, S_2^y)$ for all $(x, y) \in G \cup E \cup B$.

Given (x_i, y_j), if $(x_{i+1}, y_{i+1}) \in G \cup E \cup B$, then this pair is the *PQ-predecessor* of (x_i, y_j). Let i' be the smallest index that is larger than i such that $(x_{i'}, y_j) \in G \cup E \cup B$. This $(x_{i'}, y_j)$ is the *P-predecessor of* (x_i, y_j). Let j' be the smallest index larger than j such that $(x_i, y_{j'}) \in G \cup E \cup B$. This $(x_i, y_{j'})$ is the *Q-predecessor of* (x_i, y_j).

LEMMA 5.4.

1. *Each intersecting* $(x_i, y_j) \in (G \cup E) - B$ *has a P-predecessor* (x_{i+1}, y_j)*, a Q-predecessor* (x_i, y_{j+1})*, and a PQ-predecessor* (x_{i+1}, y_{j+1})*.*
2. *Each nonintersecting* $(x_i, y_j) \in E - B$ *has a P-predecessor* $(x_{i'}, y_j)$ *and a Q-predecessor* $(x_i, y_{j'})$*. Also,* $(P[i, i' - 1], Q[j, j' - 1])$ *is nonintersecting.*
3. *Each nonintersecting* $(x_i, y_1) \in G_P - B$ *has a P-predecessor* (x_{i+1}, y_1) *and a Q-predecessor* (x_i, y_j)*. Moreover,* $(x_i, Q[1, j - 1])$ *is nonintersecting.*
4. *Each nonintersecting* $(x_1, y_j) \in G_Q - B$ *has a P-predecessor* (x_i, y_j) *and a Q-predecessor* (x_1, y_{j+1})*. Moreover,* $(P[1, i - 1], y_j)$ *is nonintersecting.*

Proof. Statement 1 follows from the definitions of Ψ and E. The proofs of statements 3 and 4 are similar to Case 3 in the proof of statement 2 below.

As for statement 2, by the definition of B, $x_{i'}$ and $y_{j'}$ exist. To show $(P[i, i' - 1], Q[j, j' - 1])$ is nonintersecting, we consider the following four cases. The proofs of their symmetric cases are similar to theirs and are omitted for brevity.

Case 1. (x_i, y_j) is the ceiling of a nonintersecting leaf $(P[i, i_2], Q[j, j_2]) \in \Psi$.

Since (x_i, y_{j_2+1}) and (x_{i_2+1}, y_j) are in E, $i' \le i_2 + 1$ and $j' \le j_2 + 1$. Then because $(P[i, i_2], Q[j, j_2])$ is nonintersecting, so is $(P[i, i' - 1], Q[j, j' - 1])$.

Case 2. (x_i, y_j) is the Q-diagonal of a nonintersecting leaf $(P[i, i_2], Q[j_1, j-1])$ (or symmetrically, (x_i, y_j) is the P-diagonal of a nonintersecting leaf $(P[i_1, i-1], Q[j, j_2])$). Since (x_{i_2+1}, y_j) is the floor of $(P[i, i_2], Q[j_1, j-1])$, $(x_{i_2+1}, y_j) \in E$ and thus $i' \le i_2+1$. Let j'' be the smallest index such that $j \le j''$ and $(P[i, i_2], y_{j''})$ is intersecting. There are two subcases.

Case 2a. j'' does not exist. Then, $(P[i, i_2], Q[j, q'])$ is nonintersecting and therefore $(P[i, i' - 1], Q[j, j' - 1])$ is nonintersecting.

Case 2b. j'' exists. Let $Q[j_3, j_4]$ be a regular path that contains $y_{j''}$ and is of the same length as $Q[j_1, j - 1]$. Note that $j \le j_3$ and $(P[i, i_2], Q[j_3, j_4]) \in \Psi$. There are two subcases.

Case 2b(1). $j_3 = j$. Then (x_i, y_j) is the ceiling of $(P[i, i_2], Q[j_3, j_4])$. Since (x_i, y_j) is nonintersecting, it is the ceiling of a nonintersecting leaf in Ψ which is a descendant of $(P[i, i_2], Q[j_3, j_4])$. Therefore, Case 2b(1) is reduced to Case 1.

Case 2b(2). $j_3 > j$. By the construction of Ψ, $(x_i, y_{j_3}) \in E$ and thus $j' \le j_3$. By the choice of $Q[j_3, j_4]$, $(P[i, i_2], Q[j, j_3 - 1])$ is nonintersecting and so is $(P[i, i' - 1], Q[j, j' - 1])$.

Case 3. (x_i, y_j) is the Q-diagonal of an intersecting leaf (x_i, y_{j-1}) (or, symmetrically, (x_i, y_j) is the P-diagonal of an intersecting leaf (x_{i-1}, y_j)). Since $(x_{i+1}, y_j) \in E$, $i' = i + 1$ and $P[i, i' - 1] = x_i$. Let j'' be the smallest index such that $j < j''$ and $(x_i, y_{j''})$ is intersecting. There are two subcases.

Case 3a. j'' does not exist. Then, $(x_i, Q[j, q'])$ is nonintersecting and therefore $(P[i, i' - 1], Q[j, j' - 1])$ is nonintersecting.

Case 3b. j'' exists. Then, $(x_i, y_{j''}) \in E$ and $j' \le j''$. By the choice of j'', $(x_i, Q[j, j'' - 1])$ is nonintersecting. Thus, $(P[i, i' - 1], Q[j, j' - 1])$ is nonintersecting.

Case 4. (x_i, y_j) is the floor of a leaf $(P[i_1, i - 1], Q[j_1, j - 1])$, which may or may not be intersecting. Let $(P[i_3, i_4], Q[j_3, j_4])$ be the lowest ancestor of $(P[i_1, i - 1], Q[j_1, j - 1])$ in Ψ such that (x_i, y_j) is not the floor of $(P[i_3, i_4], Q[j_3, j_4])$. This ancestor exists because $(x_i, y_j) \notin B$. There are two subcases.

Case 4a. $j_3 = j_4$ and $i_3 < i_4$. Then, $P[i_1, i - 1]$ is a subpath of $P[i_3, \frac{i_3+i_4-1}{2}]$ and $i = \frac{i_3+i_4+1}{2}$. Also, $j_3 = j_1 = j - 1$. Thus, (x_i, y_j) is the Q-diagonal of $(P[i, i_4], y_{j-1}) \in \Psi$. By the construction of Ψ, (x_i, y_j) is the Q-diagonal of a leaf which is either $(P[i, i_4], y_{j-1})$ itself or its descendant. Depending on whether this leaf is nonintersecting or intersecting, Case 4a is reduced to Case 2 or 3.

Case 4b. $j_3 < j_4$ and $i_3 < i_4$. There are two subcases.

Case 4b(1). $P[i_1, i - 1] \subset P[i_3, \frac{i_3+i_4-1}{2}]$ and $Q[j_1, j - 1] \subset Q[j_3, \frac{j_3+j_4-1}{2}]$. Note that $i = \frac{i_3+i_4+1}{2}$, $j = \frac{j_3+j_4+1}{2}$, and (x_i, y_j) is the ceiling of $(P[\frac{i_3+i_4+1}{2}, i_4], Q[\frac{j_3+j_4+1}{2}, j_4]) \in \Psi$. Since (x_i, y_j) is nonintersecting, (x_i, y_j) is the ceiling of a nonintersecting leaf in Ψ which is $(P[\frac{i_3+i_4+1}{2}, i_4], Q[\frac{j_3+j_4+1}{2}, j_4])$ itself or a descendant. This reduces Case 4b(1) to Case 1.

Case 4b(2). $P[i_1, i-1] \subset P[i_3, \frac{i_3+i_4-1}{2}]$ and $Q[j_1, j-1] \subset Q[\frac{j_3+j_4+1}{2}, j_4]$ (or, symmetrically, $P[i_1, i-1] \subset P[\frac{i_3+i_4+1}{2}, i_4]$ and $Q[j_1, j-1] \subset Q[j_3, \frac{j_3+j_4-1}{2}]$). Note that $i = \frac{i_3+i_4+1}{2}$, $j = j_4 + 1$, and (x_i, y_j) is the Q-diagonal of $(P[\frac{i_3+i_4+1}{2}, i_4], Q[\frac{j_3+j_4+1}{2}, j_4]) \in \Psi$. Then, (x_i, y_j) is the Q-diagonal of a leaf which is $(P[\frac{i_3+i_4+1}{2}, i_4], Q[\frac{j_3+j_4+1}{2}, j_4])$ itself or a descendant. Depending on whether this leaf is nonintersecting or intersecting, Case 4b(2) is reduced to Case 2 or 3. □

5.4. Counting lemmas. We now give some counting lemmas that are used in section 5.6 to bound One-One's time complexity.

For all $(P[i_1, i_2], Q[j_1, j_2]) \in \Psi$,

- $C(P[i_1, i_2], Q[j_1, j_2])$ denotes the set of all ceilings of the leaves in Ψ which are either $(P[i_1, i_2], Q[j_1, j_2])$ itself or its descendants;
- $D(P[i_1, i_2], Q[j_1, j_2])$ denotes the set of all Q-diagonals of the leaves in Ψ which are either $(P[i_1, i_2], Q[j_1, j_2])$ itself or its descendants;
- $I(P[i_1, i_2], Q[j_1, j_2]) = \{(x_i, y_j) \mid x_i \in P[i_1, i_2], y_j \in Q[j_1, j_2]$ and (x_i, y_j) is intersecting$\}$.

LEMMA 5.5.

1. $|I(P[1, p' - 1], Q[1, q' - 1])| \le n$.
2. Ψ *has* $O(n \log(q + 1))$ *leaves of the form* $(P[i_1, i_2], Q[j_1, j_2])$ *where* $j_1 < j_2$.
3. Ψ *has* $O(n \log(q + 1))$ *pairs of the form* $(P[i_1, i_2], y_j)$ *where* $P[i_1, i_2]$ *is of length* $\frac{p'-1}{q'-1}$.
4. $|E| = O(n \log(p + 1))$.

Proof. Statements 1–3 are proved below. The proof of Statement 4 is similar to those of Statements 2 and 3.

Statement 1. For all distinct intersecting pairs (x_i, y_j) and $(x_{i'}, y_{j'})$, the leaf labels shared by the subtrees T_1^u where $u \in X_i$ and the subtrees T_2^v where $v \in Y_i$ are different from the shared labels for $X_{i'}$ and $Y_{j'}$. Statement 1 then follows from the fact that S_1 and S_2 share n leaf labels.

Statements 2 *and* 3. On each level of Ψ, for all distinct pairs $(P[i_1, i_2], Q[j_1, j_2])$ and $(P[i'_1, i'_2], Q[j'_1, j'_2])$, $I(P[i_1, i_2], Q[j_1, j_2]) \cap I(P[i'_1, i'_2], Q[j'_1, j'_2]) = \emptyset$. Thus, each level has at most $|I(P[1, p' - 1], Q[1, q' - 1])|$ nonleaf pairs. Consequently, from the second level downward, each level has at most $4 \cdot |I(P[1, p'-1], Q[1, q'-1])|$ pairs. These two statements then follow from Statement 1 and the fact that the pairs specified in these two statements are within the top $1 + \log(q' - 1)$ levels of Ψ. □

A pair (x_i, y_j) is *P-regular* if $[i, i' - 1]$ is a regular interval where $(x_{i'}, y_j)$ is the P-predecessor of (x_i, y_j). (We do not need the notion of Q-regular because $p' \ge q'$.)

Given a regular $[i_1, i_2]$, a set $\{h_1, \ldots, h_k\}$ *regularly partitions* $[i_1, i_2]$ if $h_1 = i_1$ and the intervals $[h_1, h_2 - 1], [h_2, h_3 - 1], \ldots, [h_{k-1}, h_k - 1], [h_k, i_2]$ are all regular.

LEMMA 5.6.

1. *Assume that* $j > 1$ *and* $P([i_1, i_2], y_j) \in \Psi$. *If the P-predecessor* (x_i, y_j) *of some* $(x_{i'}, y_j) \in C(P[i_1, i_2], y_j)$ *is not in* $\{(x_{i_2+1}, y_j)\} \cup C(P[i_1, i_2], y_j)$, *then* $P([i_1, i_2], y_{j-1}) \in \Psi$ *and* $(x_i, y_j) \in D(P[i_1, i_2], y_{j-1})$.
2. *Assume that* $j < q'$ *and* $P([i_1, i_2], y_{j-1}) \in \Psi$. *If the P-predecessor* (x_i, y_j) *of some* $(x_{i'}, y_j) \in D(P[i_1, i_2], y_{j-1})$ *is not in* $\{(x_{i_2+1}, y_j)\} \cup D(P[i_1, i_2], y_{j-1})$, *then* $P([i_1, i_2], y_j) \in \Psi$ *and* $(x_i, y_j) \in C(P[i_1, i_2], y_j)$.
3. *For every* $(P[i_1, i_2], y_j) \in \Psi$, *the set* $\{i \mid (x_i, y_j) \in C(P[i_1, i_2], y_j)\}$ *regularly partitions* $[i_1, i_2]$ *and so does the set* $\{i \mid (x_i, y_j) \in D(P[i_1, i_2], y_j)\}$.
4. *For all* $(P[i_1, i_2], y_j) \in \Psi$, *every pair in* $C(P[i_1, i_2], y_j) \cup D(P[i_1, i_2], y_j)$ *is P-regular.*
5. *At most* $O(n \log(q + 1))$ *of the nonintersecting pairs of E are P-irregular.*

Proof. The proofs of Statements 1 and 5 are detailed below. The proof of Statement 2 is similar to that of Statement 1 and is omitted. Statement 3 is obvious. Statement 4 follows from the first three statements and the fact that if two sets regularly partition $[i_1, i_2]$, then so does their union.

Statement 1. Note that $i_1 < i \le i_2$ and $q' > j > 1$. The pair (x_i, y_j) can be the ceiling, the P-diagonal, the Q-diagonal, or the floor of some leaf $(P[i_3, i_4], Q[j_3, j_4]) \in$

Ψ. These four cases are discussed below.

Case 1. (x_i, y_j) is the ceiling. Then $i = i_3$ and $j = j_3$. Since $i_1 < i \le i_2$ and both $[i, i_4]$ and $[i_1, i_2]$ are regular, $[i, i_4] \subset [i_1, i_2]$. Since the length of $P[i_1, i_2]$ is at most $\frac{p'-1}{q'-1}$, so is the length of $P[i, i_4]$. Thus $Q[j_3, j_4] = y_j$ and $(P[i, i_4], y_j)$ is a descendant of $(P[i_1, i_2], y_j)$. This contradicts the assumption that $(x_i, y_j) \notin C(P[i_1, i_2], y_j)$, and this case cannot exist.

Case 2. (x_i, y_j) is the P-diagonal. Then $i = i_4 + 1$ and $j = j_3$. As in Case 1, $Q[j_3, j_4] = y_j$ and $(P[i_3, i-1], y_j)$ is a descendant of $(P[i_1, i_2], y_j)$. Thus, there exists a leaf $(P[i, i_6], y_j)$ that is a descendant of $(P[i_1, i_2], y_j)$. Because (x_i, y_j) is the ceiling of this leaf, the existence of this leaf contradicts the assumption that $(x_i, y_j) \notin C(P[i_1, i_2], y_j)$ and this case cannot exist.

Case 3. (x_i, y_j) is the Q-diagonal. Then, $i = i_3$ and $j = j_4 + 1$. As in Case 1, $[i, i_4] \subset [i_1, i_2]$ and $Q[j_3, j_4] = y_{j-1}$. Since $(P[i, i_4], y_{j-1}) \in \Psi$, $(P[i_1, i_2], y_{j-1}) \in \Psi$. Then $(P[i, i_4], y_{j-1})$ is a descendant of $(P[i_1, i_2], y_{j-1})$ and $(x_i, y_j) \in D(P[i_1, i_2], y_{j-1})$.

Case 4. (x_i, y_j) is the floor. Then, $i = i_4 + 1$ and $j = j_4 + 1$. As in Case 3, $(P[i_1, i_2], y_{j-1}) \in \Psi$, $Q[j_3, j-1] = y_{j-1}$ and $(P[i_3, i-1], y_{j-1})$ is a descendant of $(P[i_1, i_2], y_{j-1})$. Thus, there is a leaf $(P[i, i_6], y_{j-1})$ which is a descendant of $(P[i_1, i_2], y_{j-1})$. Since (x_i, y_j) is this leaf's Q-diagonal, it is in $D(P[i_1, i_2], y_{j-1})$.

Statement 5. Note that E consists of the following three types of pairs:

1. the ceiling, diagonals, and floor of a leaf $(P[i_1, i_2], Q[j_1, j_2]) \in \Psi$ where $j_1 < j_2$;
2. the P-diagonal and floor of $(P[i_1, i_2], y_j]) \in \Psi$ where $P[i_1, i_2]$ is of length $\frac{p'-1}{q'-1}$;
3. the pairs in $C(P[i_1, i_2], j]) \cup D(P[i_1, i_2], y_j])$ where $(P[i_1, i_2], j]) \in \Psi$ and $P[i_1, i_2]$ is of length $\frac{p'-1}{q'-1}$.

By Statement 4, only the pairs of the first two types may be P-irregular. This statement then follows from Lemmas 5.5(5.5) and 5.5(5.5). ∎

5.5. Recurrences. One-One uses the following formulas to recursively compute $\text{RR}(S_1^{x_i}, S_2^{y_j})$ for $(x_i, y_j) \in G \cup E \cup B$ in terms of the RR values of the appropriate P-predecessor, Q-predecessor, and PQ-predecessor of (x_i, y_j).

For vertex subsets U of S_1 and V of S_2, $\text{M}(U, V)$ denotes the maximum weight of any matching of the bipartite graph $(U, V, U \times V)$ where the weight of an edge (u, v) is $\text{RR}(S_1^u, S_2^v)$. Let $\text{M}(U, v) = \text{M}(U, \{v\})$ and $\text{M}(u, V) = \text{M}(\{u\}, V)$. Given two vertices $x \in S_1$ and $y \in S_2$, let $\overline{\text{M}}(U, V, x, y)$ be the maximum weight of any matching of the same graph without the edge (x, y).

LEMMA 5.7. *For each* $(x_i, y_j) \in B$, $\text{RR}(S_1^x, S_2^y) = 0$.

Proof. This lemma follows from the fact that $p' > p$, $q > q$, and the new labels of S_1 and S_2 are different from one another and the labels of T_1 and T_2. ∎

FACT 2 (see [47]). *For all vertices* $u \in S_1$ *and* $v \in S_2$,

$$\text{RR}(S_1^u, S_2^v) = \max \left\{ \begin{array}{l} \text{M}(K(u, S_1), K(v, S_2)) \\ \text{M}(u, K(v, S_2)) \\ \text{M}(K(u, S_1), v) \end{array} \right\}.$$

Proof. To form maximum agreement subtrees of S_1^u and S_2^v, there are three cases. (1) $\text{M}(K(u, S_1), K(v, S_2))$ accounts for matching u to v. (2) $\text{M}(u, K(v, S_2))$ accounts for matching u to a proper descendant of v. (3) $\text{M}(K(u, S_1), v)$ accounts for matching v to a proper descendant of u. ∎

LEMMA 5.8. *For all (x_i, y_j) where $i < p'$ and $j < q'$, regardless of whether (x_i, y_j) is intersecting or nonintersecting,*

$$\textsc{rr}(S_1^{x_i}, S_2^{y_j}) = \max \left\{ \begin{array}{l} \textsc{m}(X_i, Y_j) + \textsc{rr}(S_1^{x_{i+1}}, S_2^{y_{j+1}}) \\ \overline{\textsc{m}}(X_i \cup \{x_{i+1}\}, Y_j \cup \{y_{j+1}\}, x_{i+1}, y_{j+1}) \\ \textsc{rr}(S_1^{x_i}, S_2^{y_{j+1}}) \\ \textsc{m}(x_i, Y_j) \\ \textsc{rr}(S_1^{x_{i+1}}, S_2^{y_j}) \\ \textsc{m}(X_i, y_j) \end{array} \right\}.$$

Proof. This lemma follows from Fact 2 with a finer case analysis for the cases in the proof of Fact 2. □

LEMMA 5.9. *For each nonintersecting $(x_i, y_j) \in E - B$ with P-predecessor $(x_{i'}, y_j)$ and Q-predecessor $(x_i, y_{j'})$,*

$$\textsc{rr}(S_1^{x_i}, S_2^{y_j}) = \max \left\{ \begin{array}{l} \max_{j'' \in [j, j'-1]} \textsc{m}(x_{i'}, Y_{j''}) + \max_{i'' \in [i, i'-1]} \textsc{m}(X_{i''}, y_{j'}) \\ \textsc{rr}(S_1^{x_i}, S_2^{y_{j'}}) \\ \textsc{rr}(S_1^{x_{i'}}, S_2^{y_j}) \end{array} \right\}.$$

Proof. This lemma follows from Lemma 5.4(2) and is obtained by iterative applications of Lemma 5.8. The following properties are used. Since $(P[i, i'-1], Q[j, j'-1])$ is nonintersecting, for $i'' \in [i, i'-1]$ and $j'' \in [j, j'-1]$,

- $\textsc{m}(X_{i''}, Y_{j''}) = 0$;
- $\overline{\textsc{m}}(X_{i''} \cup \{x_{i''+1}\}, Y_{j''} \cup \{y_{j''+1}\}, x_{i''+1}, y_{j''+1}) = \textsc{m}(x_{i''}, Y_{j''}) + \textsc{m}(X_{i''}, y_{j''})$;
- $\textsc{m}(x_{i''}, Y_{j''}) = \textsc{m}(x_{i'}, Y_{j''})$;
- $\textsc{m}(X_{i''}, y_{j''}) = \textsc{m}(X_{i''}, y_{j'})$. □

For brevity, the symmetric statement of the next lemma for G_Q is omitted.

LEMMA 5.10. *For all nonintersecting pairs $(x_i, y_1) \in G_P - B$ with Q-predecessor (x_i, y_j),*

$$\textsc{rr}(S_1^{x_i}, S_2^{y_1}) = \max \left\{ \begin{array}{l} \textsc{rr}(S_1^{x_i}, S_2^{y_j}) \\ \textsc{rr}(S_1^{x_{i+1}}, S_2^{y_1}) \\ \textsc{m}(X_i, y_j) + \max_{j' \in [1, j-1]} \textsc{m}(x_{i+1}, Y_{j'}) \end{array} \right\}.$$

Proof. The proof is similar to the proof of Lemma 5.9 and follows from Lemma 5.4(3). □

5.6. The algorithm for Problem 1. We combine the discussion in sections 5.3–5.5 to give the following algorithm to solve Problem 1.

ALGORITHM ONE-ONE

begin

1. Compute S_1, S_2, P', Q', $\textsc{rp}(S_1^u, S_2, Q')$ for $u \in K(P', S_1)$, and $\textsc{rp}(S_2^v, S_1, P')$ $v \in K(Q', S_2)$;
2. Compute $G \cup E \cup B$, B, $I(P[1, p'-1], Q[1, q'-1]) - B$, the set of all nonintersecting pairs in $E - B$, and the sets of nonintersecting pairs in $G_P - B$ and $G_Q - B$, respectively;
3. Compute the following predecessors:
   - the P-predecessor, Q-predecessor, and PQ-predecessor of each pair in $I(P[1, p'-1], Q[1, q'-1]) - B$;
   - the P-predecessor and Q-predecessor of each nonintersecting pair in $E - B$;

- the Q-predecessor of each nonintersecting pair in $G_P - B$ and the P-predecessor of each nonintersecting pair in $G_Q - B$;

4. For all pairs in $G \cup E \cup B$, compute the non-RR terms in the appropriate recurrence formulas of section 5.5:
   - Lemma 5.7 for B;
   - Lemma 5.8 for $I(P[1, p'-1], Q[1, q'-1]) - B$;
   - Lemma 5.9 for the nonintersecting pairs in $E - B$;
   - Lemma 5.10 for the nonintersecting pairs in $G_P - B$ and its symmetric statement for the nonintersecting pairs in $G_Q - B$;
5. Compute the $\text{RR}(S_1^{x_i}, S_2^{y_j})$ for all $(x_i, y_j) \in G \cup E \cup B$ using the appropriate recurrence formulas given in section 5.5 and the non-RR terms computed at step 4;
6. Compute the output as follows:
   - For all $y_j \in Q$, $\text{RP}(T_1, T_2, Q)(y_j) \leftarrow \text{RR}(S_1^{x_1}, S_2^{y_j})$;
   - For all $x_i \in P$, $\text{RP}(T_2, T_1, P)(x_i) \leftarrow \text{RR}(S_1^{x_i}, S_2^{y_1})$;
   - For every $v \in K(Q, T_2)$, $\text{RP}(T_1, T_2, Q)(v) \leftarrow \text{RP}(T_2^v, T_1, P)(h)$ where h is the root of $T_1 || T_2^v$;
   - For every $u \in K(P, T_1)$, $\text{RP}(T_2, T_1, P)(u) \leftarrow \text{RP}(T_1^u, T_2, Q)(h)$ where h is the root of $T_2 || T_1^u$;

end.

To analyze One-One, we first focus on step 4. The recurrences of section 5.5 contain only four types of non-RR terms other than the constant 0 in Lemma 5.7:

1. $\text{M}(X_i, y_j)$ and $\text{M}(x_i, Y_j)$;
2. $\max_{i \in [i_1, i_2]} \text{M}(X_i, y_j)$ and $\max_{j \in [j_1, j_2]} \text{M}(x_i, Y_j)$;
3. $\text{M}(X_i, Y_j)$;
4. $\overline{\text{M}}(X_i \cup \{x_{i+1}\}, Y_j \cup \{y_{j+1}\}, x_{i+1}, y_{j+1})$.

It is important to notice that these non-RR terms can be simultaneously evaluated. In light of this observation, we compute these terms by using the techniques of section 5.1 to process the normal sequences A_i, A_u, B_j, B_v defined below:

- $A_i(j) = \text{M}(X_i, y_j)$ for all x_i and y_j;
- $B_j(i) = \text{M}(x_i, Y_j)$ for all y_j and x_i;
- $A_u(j) = \text{RR}(S_1^u, S_2^{y_j})$ for all $u \in K(P', S_1)$ and y_j;
- $B_v(i) = \text{RR}(S_2^v, S_1^{x_i})$ for all $v \in K(Q', S_2)$ and x_i.

Note that A_i and A_u have length q', and A_i is the joint of all A_u where $u \in X_i$. Similarly, B_j and B_v have length p', and B_j is the joint of all B_v where $v \in Y_j$.

LEMMA 5.11.

1. *The minimal condensed forms of the sequences A_u and B_v have a total size of $O(n)$ and can be computed in $O(n)$ time.*
2. *The minimal condensed forms of the sequences A_i and B_j have a total size of $O(n)$ and can be computed in $O(n)$ time.*

Proof. Statement 2 follows from Statement 1 and Lemma 5.1. Below we only prove Statement 1 for A_u; Statement 1 for B_v is similarly proved. We first compute a condensed form $\overline{A}_u$ for each A_u as follows:

1. For all $u \in K(P', S_1)$, compute $S_{2,u} = S_2 || S_1^u$ and $Q_u = Q' || S_1^u$.
2. For all u where Q_u is nonempty, do the following steps:
   (a) $\overline{A}_u \leftarrow \{(j, w) \mid y_j \in Q_u, w = \text{RP}(S_1^u, S_2, Q')(y_j)\}$.
   (b) Compute all tuples $(\hat{v}, v, y_j)$ where $\hat{v} \in K(Q_u, S_{2,u})$, $v \in K(Q', S_2)$, $\hat{v} \in S_2^v$, and $v \in Y_j$.
   (c) Find the smallest s such that some $(\hat{v}, v, y_s)$ is obtained at Step 2(b).

(d) If there is only one $(\hat{v}, v, y_s)$, then add to $\overline{A}_u$ the pair (s, w) where $w = \text{RP}(S_1^u, S_2, Q')(\hat{v})$.

3. For all u where $S_{2,u}$ is nonempty and Q_u is empty, do the following steps:
   (a) Compute $\hat{v}$, v, and y_s where $\hat{v}$ is the root of $S_{2,u}$, $v \in K(Q', S_2)$, $\hat{v} \in S_2^v$, and $v \in Y_s$.
   (b) $\overline{A}_u \leftarrow \{(s, w)\}$, where $w = \text{RP}(S_1^u, S_2, Q')(\hat{v})$.
4. For all u where $S_{2,u}$ is empty, $\overline{A}_u \leftarrow \emptyset$.

The correctness proof of this algorithm has three cases.

Case 1. Q_u is nonempty. Let $y_{j_1}, y_{j_2}, \ldots, y_{j_k} = Q_u$. Let $j_0 = 0$. Then, for all $k' \in [1, k]$ and all $j \in [j_{k'-1} + 1, j_{k'}]$, $S_2^{y_j} || S_1^u = S_{2,u}^{y_{k'}}$ and by Lemma 3.1, $A_u(j) = \text{RP}(S_1^u, S_2, Q')(y_{k'})$. There are two subcases for $j > j_k$.

Case 1a. Step 2(b) finds two or more $(\hat{v}, v, y_s)$. Then $y_s \in Q_u$, $s = j_k$, and for all $j \in [j_k + 1, q']$, $S_2^{y_j} || S_2^u$ is empty and $A_u(j) = 0$.

Case 1b. Step 2(b) finds only one $(\hat{v}, v, y_s)$. Then $y_s \notin Q_u$ and $s > j_k$. For all $j \in [j_k + 1, s]$, $S_2^{y_j} || S_1^u = S_{2,u}^{\hat{v}}$ and $A_u(j) = \text{RP}(S_1^u, S_2, Q')(\hat{v})$. For all $j \in [s + 1, q']$, $S_2^{y_j} || S_1^u$ is empty and $A_u(j) = 0$.

Thus, the $\overline{A}_u$ of Step 2 is a condensed form of A_u for Case 1.

Case 2. $S_{2,u}$ is nonempty and Q_u is empty. This case is similar to Case 1b, and Step 3 computes a correct condensed form $\overline{A}_u$ for this case.

Case 3. $S_{2,u}$ is empty. This case is obvious, and Step 4 correctly computes a condensed form $\overline{A}_u$ of A_u for this case.

The total size of all $\overline{A}_u$ is at most that of the RP mappings of S_1, S_2, P', and Q', which is the desired $O(n)$. Step 1 takes $O(n)$ time using Fact 1. The other steps can be implemented in $O(n)$ time in straightforward manners using radix sort and tree traversal. As discussed in section 3, the RP mappings are evaluated by radix sort. Once the forms $\overline{A}_u$ are obtained, we can in $O(n)$ time radix sort the pairs in all $\overline{A}_u$ and then delete all unnecessary pairs to obtain the desired minimal condensed forms. □

LEMMA 5.12. *All the non-*RR *terms of the first two types for the pairs in* $G \cup E \cup B$ *can be evaluated in* $O(n \log(p+1) \log(q+1))$ *time.*

Proof. The value of $\text{M}(X_i, y_j)$ is that of the point query $([i, i], j)$ for $A_1, \ldots, A_{q'}$, and the value of $\max_{i \in [i_1, i_2]} \text{M}(X_i, y_j)$ is that of the interval query $([i_1, i_2], j)$. By Lemma 5.5(5.5), there are $O(n \log(p+1))$ such terms required for the pairs in $G \cup E \cup B$. Given the results of steps 2 and 3 of One-One, we can determine all such terms and the corresponding queries in $O(n \log(p+1))$ time. By Lemma 5.6(5), only $O(n \log(q+1))$ of these queries are not P-regular. By Lemmas 5.11(2) and 5.2(2), we can evaluate these queries in $O(n \log(p+1) \log(q+1))$ time. The terms $\text{M}(x_i, Y_j)$ and $\max_{j \in [j_1, j_2]} \text{M}(x_i, Y_j)$ are similarly evaluated in $O(n \log(p+1) \log(q+1))$ time. The analysis for these terms is easier because $p' \geq q'$, and it does not involve the notion of Q-regularity. □

LEMMA 5.13. *The non-*RR *terms of the third and the fourth type for the pairs in* $G \cup E \cup B$ *can be evaluated within the following time complexity:*

1. $O(nd \log d)$ *or alternatively* $O(n\sqrt{d} \log n)$ *for the third type*;
2. $O(nd^2 \log d)$ *or alternatively* $O(nd\sqrt{d} \log n)$ *for the fourth type.*

Proof. To prove Statement 1, we consider the graphs $(X_i, Y_j, X_i \times Y_j)$ on which the desired terms $\text{M}(X_i, Y_j)$ are defined. Let $Z_{i,j}$ be the subgraph of $(X_i, Y_j, X_i \times Y_j)$ constructed by removing all zero-weight edges and all resulting isolated vertices. The edges of $Z_{i,j}$ are computed as follows:

1. For all $u \in K(P', S_1)$, compute $S_{2,u} = S_2 || S_1^u$ and $Q_u = Q' || S_1^u$.
2. For all $S_{2,u}$ is nonempty, do the following steps:

(a) If Q_u is nonempty, compute all tuples (u, v, w) where $\hat{v} \in K(Q_u, S_{2,u})$, $v \in K(Q', S_2)$, $\hat{v} \in S_2^v$, and $w = \text{RP}(S_1^u, S_2, Q')(\hat{v})$.
(b) If Q_u is empty, compute the tuple (u, v, w) where $\hat{v}$ is the root of $S_{2,u}$, $v \in K(Q', S_2)$, $\hat{v} \in S_2^v$, and $w = \text{RP}(S_1^u, S_2, Q')(\hat{v})$.

This algorithm captures all the nonzero-weight (u, v). At step 2, $S_{2,u}^{\hat{v}} = S_2^v || S_1^u$ and by Lemma 3.1 $\text{RR}(S_1^u, S_2^v) = \text{RP}(S_1^u, S_2, Q')(\hat{v})$. Thus, the first two components of the obtained tuples form the edges of all desired $Z_{i,j}$ and the third components are the weights of these edges. We use Fact 1 to implement step 1 in $O(n)$ time. We can implement step 2 in $O(n)$ time using radix sort and tree traversal. Note that step 2 uses radix sort to evaluate RP mappings. With the tuples (u, v, w) obtained, we use radix sort to construct all desired $Z_{i,j}$ in $O(n)$ time. Let $m_{i,j}$ and $n_{i,j}$ be the numbers of edges and vertices in $Z_{i,j}$, respectively. Since an edge weighs at most n, we can compute $\text{M}(X_i, Y_j)$ in $O(n_{i,j} \cdot m_{i,j} + n_{i,j}^2 \cdot \log n_{i,j})$ and alternatively in $O(m_{i,j} \cdot \sqrt{n_{i,j}} \cdot \log(n \cdot n_{i,j}))$ time [21, 42]. Statement 1 then follows from the fact that $n_{i,j} \leq 2d'$, $n_{i,j} \leq 2m_{i,j}$, and by Lemma 5.5(5.5) the sum of all $m_{i,j}$ is at most n.

To prove Statement 2, we similarly process the bipartite graphs on which the desired terms $\overline{\text{M}}(X_i \cup \{x_{i+1}\}, Y_j \cup \{y_{j+1}\}, x_{i+1}, y_{j+1})$ are defined. The key difference from the third type is that in addition to some of the edges in $Z_{i,j}$, we need certain nonzero-weight (u, y_{j+1}) for $u \in X_i$ and (x_{i+1}, v) for $v \in Y_j$. Since these edges are required only for intersecting (x_i, y_j), by Lemma 5.5(5.5), $O(dn)$ such edges are needed. We use Lemma 5.11(1) to compute the weights of these edges in $O(dn)$ time. Due to these edges, the total time complexity for the fourth type is $O(d)$ times that for the third type. □

The next theorem serves to prove Theorem 4.1 given at the beginning of section 4.

THEOREM 5.14. *One-One solves Problem* 1 *with the following time complexities:*

$$O(nd^2 \log d + n \log(p+1) \log(q+1))$$

or, alternatively,

$$O(nd\sqrt{d} \log n + n \log(p+1) \log(q+1)).$$

Proof. The correctness of One-One follows from Lemma 5.3 and sections 5.3–5.5. As for the time complexity, step 1 is obvious and takes $O(n)$ time. By computing Ψ, we can compute the sets E and $I(P[1, p'-1], Q[1, q'-1])$. Since the leaf labels of S_1 and S_2 are from $[1, O(n)]$, each level of Ψ can be computed in $O(n)$ time. Since Ψ has $O(\log(p+1))$ levels, E and $I(P[1, p'-1], Q[1, q'-1])$ can be computed in $O(n \log(p+1))$ time. With these two sets obtained, we can compute all the desired sets in $O(n \log(p+1))$ time. Thus, step 2 takes $O(n \log(p+1))$ time. Step 3 takes $O(n \log(p+1))$ time using radix sort. The time complexity of step 4 dominates that of One-One. This step uses Lemmas 5.12 and 5.13 and takes $O(n \log(p+1) \log(q+1) + nd^2 \log d)$ time or, alternatively, $O(n \log(p+1) \log(q+1) + nd\sqrt{d} \log n)$ time. Step 5 spends $O(n \log(p+1))$ time using radix sort to create pointers from the pairs in $G \cup E \cup B$ to appropriate predecessors. Step 5 then takes $O(1)$ time per pair in $G \cup E \cup B$ and $O(n \log(p+1))$ time in total. Step 6 takes $O(n \log(p+1))$ time. It uses radix sort to access the desired RR values and evaluate the input mappings. It also uses Fact 1 to compute all $T_1 || T_2^v$ and $T_2 || T_1^u$. □

6. Discussions. We answer the main problem of this paper with the following theorem and conclude with an open problem.

THEOREM 6.1. *Let T_1 and T_2 be two evolutionary trees with n leaves each. Let d be their maximum degree. Given T_1 and T_2, a maximum agreement subtree of T_1 and T_2 can be computed in $O(nd^2 \log d \log^2 n)$ time or alternatively in $O(nd\sqrt{d} \log^3 n)$ time. Thus, if d is bounded by a constant, a maximum agreement subtree can be computed in $O(n \log^2 n)$ time.*

Proof. By Theorem 4.5, the algorithms in sections 4–5 compute $\mathrm{RR}(T_1, T_2)$ within the desired time bounds. With straightforward modifications, these algorithms can compute a maximum agreement subtree within the same time bounds. □

The next lemma establishes a reduction from the longest common subsequence problem to that of computing a maximum agreement subtree.

LEMMA 6.2. *Let $M_1 = x_1, \ldots, x_n$ and $M_2 = y_1, \ldots, y_n$ be two sequences. Assume that the symbols x_i are all distinct and so are the symbols y_j. Then, the problem of computing a longest common subsequence of M_1 and M_2 can be reduced in linear time to that of computing a maximum agreement subtree of two binary evolutionary trees.*

Proof. Given M_1 and M_2, we construct two binary evolutionary trees T_1 and T_2 as follows. Let z_1 and z_2 be two distinct symbols different from all x_i and y_i. Next, we construct two paths $P_1 = u_1, \ldots, u_{n+1}$ and $P_2 = v_1, \ldots, v_{n+1}$. T_1 is formed by making u_1 the root, attaching x_i to u_i as a leaf, and attaching z_1 and z_2 to u_{n+1} as leaves. Symmetrically, T_2 is formed by making v_1 the root, attaching y_i to v_i, and attaching z_1 and z_2 to v_{n+1}. The lemma follows from the straightforward one-to-one onto correspondence between the longest common subsequences of M_1 and M_2 and the maximum agreement subtrees of T_1 and T_2. □

We can use Lemma 6.2 to derive lower complexity bounds for computing a maximum agreement subtree from known bounds for the longest common subsequence problem in various models of computation [3, 6, 23, 29, 30, 32, 50]. This paper assumes a comparison model where two labels x and y can be compared to determine whether x is smaller than y or $x = y$ or x is greater than y. Since the longest common subsequence problem in Lemma 6.2 requires $\Omega(n \log n)$ time in this model [31], the same bound holds for the problem of computing a maximum agreement subtree of two evolutionary trees where d is bounded by a constant. It would be significant to close the gap between this lower bound and the upper bound of $O(n \log^2 n)$ stated in Theorem 6.1. Recently, Farach, Przytycka, and Thorup [13] independently developed an algorithm that runs in $O(n\sqrt{d} \log^3 n)$ time. For binary trees, Cole and Hariharan [8] gave an $O(n \log n)$-time algorithm. It may be possible to close the gap by incorporating ideas used in those two results and this paper.

Acknowledgments. The author is deeply appreciative for the extremely thorough and useful suggestions given by the anonymous referee. The author thanks Joseph Cheriyan, Harold Gabow, Andrew Goldberg, Dan Gusfield, Dan Hirschberg, Phil Klein, Phil Long, K. Subramani, Bob Tarjan, and Tandy Warnow for helpful comments, discussions, and references.

REFERENCES

[1] K. ABRAHAMSON, N. DADOUN, D. G. KIRKPATRICK, AND T. PRZYTYCKA, *A simple tree contraction algorithm*, J. Algorithms, 10 (1989), pp. 287–302.

[2] R. AGARWALA AND D. FERNÁNDEZ-BACA, *A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed*, SIAM J. Comput., 23 (1994), pp. 1216–1224.

[3] A. V. AHO, D. S. HIRSCHBERG, AND J. D. ULLMAN, *Bounds on the complexity of the longest common subsequence problem*, J. Assoc. Comput. Mach., 23 (1976), pp. 1–12.

[4] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison–Wesley, Reading, MA, 1974.
[5] A. V. Aho, Y. Savig, T. G. Szymanski, and J. D. Ullman, *Inferring a tree from the lowest common ancestors with an application to the optimization of relational expressions*, SIAM J. Comput., 10 (1981), pp. 405–421.
[6] A. Apostolico and C. Guerra, *The longest common subsequence problem revisited*, Algorithmica, 2 (1987), pp. 315–336.
[7] H. L. Bodlaender, M. R. Fellows, and T. J. Warnow, *Two strikes against perfect phylogeny*, in Lecture Notes in Computer Science 623: Proceedings of the 19th International Colloquium on Automata, Languages, and Programming, Springer-Verlag, New York, 1992, pp. 273–283.
[8] R. Cole and R. Hariharan, *An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees*, in Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, 1996, pp. 323–332.
[9] T. H. Cormen, C. L. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1991.
[10] W. H. E. Day and D. Sankoff, *Computational complexity of inferring phylogenies from chromosome inversion data*, J. Theoret. Biology, 124 (1987), pp. 213–218.
[11] S. Dress and M. Steel, *Convex tree realizations of partitions*, Appl. Math. Lett., 5 (1992), pp. 3–6.
[12] M. Farach, S. Kannan, and T. Warnow, *A robust model for finding optimal evolutionary trees*, Algorithmica, 13 (1995), pp. 155–179.
[13] M. Farach, T. M. Przytycka, and M. Thorup, *Computing the agreement of trees with bounded degrees*, in Lecture Notes in Computer Science 979: Proceedings of the Third Annual European Symposium on Algorithms, P. Spirakis, ed., Springer-Verlag, New York, 1995, pp. 381–393.
[14] M. Farach and M. Thorup, *Fast comparison of evolutionary trees (extended abstract)*, in Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, 1994, pp. 481–488.
[15] M. Farach and M. Thorup, *Optimal evolutionary tree comparison by sparse dynamic programming (extended abstract)*, in Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science, 1994, pp. 770–779.
[16] J. Felsenstein, *Numerical methods for inferring evolutionary trees*, The Quarterly Review of Biology, 57 (1982), pp. 379–404.
[17] J. Felsenstein, *Inferring evolutionary trees from DNA sequences*, in Statistical Analysis of DNA Sequence Data, B. Weir, ed., Marcel Dekker, New York, 1983, pp. 133–150.
[18] J. Felsenstein, *Phylogenies from molecular sequences: Inference and reliability*, Ann. Rev. Genetics, 22 (1988), pp. 521–565.
[19] C. R. Finden and A. D. Gordon, *Obtaining common pruned trees*, J. Classification, 2 (1985), pp. 255–276.
[20] A. Friday, *Quantitative aspects of the estimation of evolutionary trees*, Folia Primatologica, 53 (1989), pp. 221–234.
[21] H. N. Gabow and R. E. Tarjan, *Faster scaling algorithms for network problems*, SIAM J. Comput., 18 (1989), pp. 1013–1036.
[22] H. Gazit, G. L. Miller, and S. H. Teng, *Optimal tree contraction in the EREW model*, in Concurrent Computations: Algorithms, Architecture, and Technology, S. T. and B.W. Dickinson and S. Schwartz, eds., Plenum, New York, 1988, pp. 139–156.
[23] R. N. Goldberg, *Minimal string difference encodings*, J. Algorithms, 3 (1982), pp. 147–156.
[24] D. Gusfield, *Efficient algorithms for inferring evolutionary trees*, Networks, 21 (1991), pp. 19–28.
[25] D. Harel and R. E. Tarjan, *Fast algorithms for finding nearest common ancestors*, SIAM J. Comput., 13 (1984), pp. 338–355.
[26] J. J. Hein, *An optimal algorithm to reconstruct trees from additive distance data*, Bull. Math. Biology, 51 (1989), pp. 597–603.
[27] M. D. Hendy, *The relationship between simple evolutionary tree models and observable sequence data*, Systematic Zoology, 38 (1989), pp. 310–321.
[28] M. D. Hendy and D. Penny, *Branch and bound algorithms to determine minimal evolutionary trees*, Math. Biosciences, 59 (1982), pp. 277–290.
[29] D. S. Hirschberg, *A linear space algorithm for computing maximal common subsequences*, Comm. Assoc. Comput. Mach., 18 (1975), pp. 341–343.
[30] D. S. Hirschberg, *Algorithms for the longest common subsequence problem*, J. Assoc. Comput. Mach., 24 (1977), pp. 664–675.

[31] D. S. Hirschberg, *An information theoretic lower bound for the longest common subsequence problem*, Inform. Proc. Lett., 7 (1978), pp. 40–41.
[32] J. W. Hunt and T. G. Szymanski, *A fast algorithm for computing longest common subsequences*, Comm. Assoc. Comput. Mach., 20 (1977), pp. 350–353.
[33] T. Jiang, E. L. Lawler, and L. Wang, *Aligning sequences via an evolutionary tree: Complexity and approximation*, in Proceedings of the 26th Annual ACM Symposium on Theory of Computing, 1994, pp. 760–769.
[34] S. K. Kannan, E. L. Lawler, and T. J. Warnow, *Determining the evolutionary tree using experiments*, J. Algorithms, 21 (1996), pp. 26–50.
[35] S. K. Kannan and T. J. Warnow, *Inferring evolutionary history from DNA sequences*, SIAM J. Comput., 23 (1994), pp. 713–737.
[36] D. Keselman and A. Amir, *Maximum agreement subtree in a set of evolutionary trees – Metrics and efficient algorithms*, in Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science, 1994, pp. 758–769; SIAM J. Comput., 26 (1997), pp. 1656–1669.
[37] L. C. Klotz and R. L. Blanken, *A practical method for calculating evolutionary trees from sequence data*, J. Theoretical Biology, 91 (1981), pp. 261–272.
[38] S. R. Kosaraju and A. L. Delcher, *Optimal parallel evaluation of tree-structured computations by raking*, in Lecture Notes in Computer Science 319: VLSI Algorithms and Architectures, the 3rd Aegean Workshop on Computing, J. H. Reif, ed., Springer-Verlag, New York, 1988, pp. 101–110.
[39] E. Kubicka, G. Kubicki, and F. McMorris, *An algorithm to find agreement subtrees*, J. Classification, 12 (1995), pp. 91–99.
[40] G. L. Miller and J. H. Reif, *Parallel tree contraction, part 1: Fundamentals*, in Advances in Computing Research: Randomness and Computation, Vol. 5, S. Micali, ed., JAI Press, Greenwich, CT, 1989, pp. 47–72.
[41] G. L. Miller and J. H. Reif, *Parallel tree contraction part 2: Further applications*, SIAM J. Comput., 20 (1991), pp. 1128–1147.
[42] J. B. Orlin and R. K. Ahuja, *New scaling algorithms for the assignment and minimum mean cycle problems*, Math. Programming, 54 (1992), pp. 41–56.
[43] D. Penny and M. Hendy, *Estimating the reliability of evolutionary trees*, Molecular Biology and Evolution, 3 (1986), pp. 403–417.
[44] A. Rzhetsky and M. Nei, *A simple method for estimating and testing minimum-evolution trees*, Molecular Biology and Evolution, 9 (1992), pp. 945–967.
[45] B. Schieber and U. Vishkin, *On finding lowest common ancestors: Simplification and parallelization*, SIAM J. Comput., 17 (1988), pp. 1253–1262.
[46] M. Steel, *The complexity of reconstructing trees from qualitative characters and subtrees*, J. Classification, 9 (1992), pp. 91–116.
[47] M. Steel and T. Warnow, *Kaikoura tree theorems: Computing the maximum agreement subtree*, Inform. Process. Lett., 48 (1993), pp. 77–82.
[48] L. Wang, T. Jiang, and E. Lawler, *Approximation algorithms for tree alignment with a given phylogeny*, Algorithmica, 16 (1996), pp. 302–315.
[49] T. J. Warnow, *Tree compatibility and inferring evolutionary history*, J. Algorithms, 16 (1994), pp. 388–407.
[50] C. K. Wong and A. K. Chandra, *Bounds for the string editing problem*, J. Assoc. Comput. Mach., 23 (1976), pp. 13–16.

SIAM J. COMPUT.
Vol. 27, No. 6, pp. 1617–1636, December 1998
© 1998 Society for Industrial and Applied Mathematics
006

OPTIMAL PREDICTION FOR PREFETCHING IN THE WORST CASE*

P. KRISHNAN† AND JEFFREY SCOTT VITTER‡

Abstract. Response time delays caused by I/O are a major problem in many systems and database applications. Prefetching and cache replacement methods are attracting renewed attention because of their success in avoiding costly I/Os. Prefetching can be looked upon as a type of online sequential prediction, where the predictions must be accurate as well as made in a computationally efficient way. Unlike other online problems, prefetching cannot admit a competitive analysis, since the optimal offline prefetcher incurs no cost when it knows the future page requests. Previous analytical work on prefetching [*J. Assoc. Comput. Mach.*, 143 (1996), pp. 771–793] consisted of modeling the user as a probabilistic Markov source.

In this paper, we look at the much stronger form of worst-case analysis and derive a randomized algorithm for pure prefetching. We compare our algorithm for every page request sequence with the important class of finite state prefetchers, making no assumptions as to how the sequence of page requests is generated. We prove analytically that the fault rate of our online prefetching algorithm converges almost surely for every page request sequence to the fault rate of the optimal finite state prefetcher for the sequence. This analysis model can be looked upon as a generalization of the competitive framework, in that it compares an online algorithm in a worst-case manner over all sequences with a powerful yet nonclairvoyant opponent. We simultaneously achieve the computational goal of implementing our prefetcher in optimal constant expected time per prefetched page using the optimal dynamic discrete random variate generator of Matias, Vitter, and Ni [*Proc. 4th Annual SIAM/ACM Symposium on Discrete Algorithms*, Austin, TX, January 1993].

Key words. caching, prefetching, competitive analysis, finite state prefetchers, response time, fault rate, hypertext, operating systems, databases, prediction, machine learning

AMS subject classifications. 68Q25, 68T05, 68P20, 68N25, 60J20

PII. S0097539794261817

1. Introduction.

Most computer memories are organized hierarchically. A typical two-level memory consists of a relatively small but fast *cache* (such as internal memory) and a relatively large but slow memory (such as disk storage). Two-level memories can also model on-chip versus off-chip memory in VLSI systems. The pages requested by an application must be in cache before computation can proceed. In the event that a requested page is not in cache, a *page fault* occurs and the application has to wait while the page is fetched from slow memory to cache. The method of fetching pages into cache only when a fault occurs is called *demand fetching*. The problem of *cache replacement* is to decide which pages to remove from cache to accommodate the incoming pages.

In many systems and database applications, users spend a significant amount of

*Received by the editors January 19, 1994; accepted for publication (in revised form) August 26, 1996; published electronically June 3, 1998. An extended abstract appears in the *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, Arlington, VA, January 1994, pp. 392–401.

http://www.siam.org/journals/sicomp/27-6/26181.html

†Bell Laboratories, 101 Crawfords Corner Road, Holmdel, NJ 07733-3030 (pk@research.bell-labs.com). Support was provided in part by the Office of Naval Research and the Defense Advanced Research Projects Agency under contract N00014–91–J–4052, ARPA order 8225, and by Air Force Office of Scientific Research grants F49620–92–J–0515 and F49620–94–1–0217. This work was done while the author was associated with Brown University and Duke University.

‡Dept. of Computer Science, Duke University, Durham, NC 27708–0129 (jsv@cs.duke.edu). Support was provided in part by National Science Foundation research grants CCR–9007851 and CCR–9522047, by Air Force Office of Scientific Research grants F49620–92–J–0515 and F49620–94–1–0217, and by a Universities Space Research Association/CESDIS associate membership.

time processing a page, and the computer and I/O system are typically idle during that period. If the computer can predict which page the user will request next, it can fetch that page into cache (if it is not already in cache) *before* the user asks for it. When the user requests the page, it is available in cache, and the user perceives a faster response time. This method of getting pages into cache in the background before they are requested is called *prefetching*.

In many hypertext and interactive database systems, there is often sufficient time between user requests to prefetch as many pages as wanted, limited only by the cache size k. We refer to prefetching under this assumption as *pure prefetching*, and we restrict our analysis to pure prefetching in this paper. Pure prefetching is an important theoretical and practical model that helps in analyzing the benefits of fetching pages in the background.

In general applications, other issues come into play. For example, prefetches are often done well in advance of when the page is expected to be needed, to take into account latency [29, 31]. User requests can also preempt prefetch requests, resulting in fewer than k prefetches being done at a time. In such situations, which we call *nonpure prefetching*, issues of cache replacement come into play; the algorithm must determine not only which page(s) to prefetch but also which page(s) to evict from cache to make room. Pure prefetchers can be converted into efficient and practical nonpure prefetchers by melding them with good cache replacement strategies. In [10], pure prefetchers are used with the popular least recently used (LRU) cache replacement strategy, and significant reductions in page fault rate (number of page faults divided by the number of page requests) are demonstrated. We expect that better pure prefetchers (e.g., the one developed in this paper) melded with better cache replacement strategies (e.g., [5, 13, 19, 20]) may yield even more impressive performance improvements.

An algorithm is *online* if it must make its decisions based only on the past history. An *offline* algorithm can use knowledge of the future. If the program generating page requests is known a priori, prefetching decisions could be made offline, as is done in compiler-directed prefetching [6, 29, 31] where prefetch instructions are explicitly inserted into the code. Without a priori knowledge or statistics of the user request pattern, as is the case in many hypertext and interactive applications (e.g., the world wide web), an algorithm for cache replacement or prefetching must be online. An important computational requirement of online prefetching (and online demand fetching) algorithms is that the time spent deciding which pages to fetch into (or evict from) cache must be minimal. In this paper, we study online pure prefetching.

1.1. Analysis technique. The notion of *competitiveness* introduced by Sleator and Tarjan [34] evaluates an online algorithm by comparing its performance with that of offline algorithms. Competitive algorithms for cache replacement are well examined in the literature [5, 13, 27, 34]. It is unreasonable to expect prefetching algorithms to be competitive in this sense. The trivial optimal offline algorithm for prefetching never faults, if it can prefetch at least one page every time. In order to be competitive, an online algorithm would have to be an almost perfect predictor for any sequence, which seems intuitively impossible. Some restrictions on the power of the offline algorithm are therefore needed for a meaningful analysis.

Vitter and Krishnan [36] analyzed pure prefetching using a form of the competitive philosophy; they assumed that the sequence of page requests was generated by a probabilistic Markov source [14]. They showed that the prediction techniques inherent in data compression methods (such as the Lempel–Ziv algorithm [37]) can be

used to get optimal pure prefetchers. Cache replacement has been studied by Karlin, Phillips, and Raghavan [20] under a different stochastic version of the competitive framework; the sequence of page requests was assumed to be generated by a Markov chain (a subset of Markov sources). In [20, 36], the online prefetching or cache replacement algorithm is compared with the optimal online algorithm that has full prior knowledge of the source. A PAC learning framework incorporating Markov sources of examples was developed in [1]. Recent empirical work on prefetching includes a pattern matching approach to prediction [30], computing various first-order statistics for prediction [32], a growing-order Markov predictor [24], prefetching in a parallel environment [22], and research projects at a lower level of abstraction including compiler-directed prefetching [6, 29, 31].

In this paper, we develop a randomized algorithm for pure prefetching and show its optimality in the limit under the following analysis strategy. Putting *no* restrictions on the generator of page requests, we compare the page fault rate of our prefetcher *for every sequence of page requests* with that of the *best finite state prefetcher for the sequence.* We also show how to implement each prefetch in *constant expected time,* independent of the number of pages in the database and the cache size, which is optimal.

The analysis strategy used in this paper is much stronger than the ones in [20, 36] for the following reasons. First, our comparison is for all sequences, without any assumption about how the sequences are generated. Second, although the comparison in [20, 36] is against the optimal computationally unlimited online algorithm with full a priori knowledge of the source, it is the case for prefetching and cache replacement that when the page request sequences are generated by a finite state Markov source (or a Markov chain), the optimal online algorithm is finite state. (See [36, Definition 4] and [20, Theorem 2].) In particular, the fault rate of the prefetcher we develop in this paper is asymptotically the same as the fault rate of the optimal prefetcher from [36] when the source is finite state Markov.

Pure prefetching can be looked upon as the following prediction problem: given an arbitrary alphabet of size α (the set of α pages in the database) and a sequence of (page) requests drawn from this alphabet, at each time instant we have to predict the best k choices for the k pages to prefetch into cache. Randomness is required in order for a predictor or prefetcher to be optimal for every sequence when compared with finite state prefetchers (FSPs) [7]. (Also see Appendix A.) In the fields of information theory and statistics [4, 8, 12, 17] interesting algorithms for binary sequences (corresponding to an alphabet size of $\alpha = 2$ pages) that make one prediction for the next page (corresponding to cache size $k = 1$) have been developed independently of [36], and the comparison is made with the best finite state predictor. However, the $\alpha = 2$, $k = 1$ case is clearly unsuitable for our prefetching scenario. The procedure in [17] may possibly be generalizable to the arbitrary alphabet case $\alpha \geq 2$ for cache size $k = 1$, but it cannot possibly make a prediction in constant time independent of α, and the $k > 1$ case is open. In [28], predictors are developed for various continuous loss functions, but they are not relevant to the harder-to-analyze discontinuous 0–1 loss functions associated with cache replacement and prefetching. The solution to the general case thus requires a fundamentally different approach from those mentioned above.

1.2. This paper. Our major contribution in this paper is a randomized algorithm for pure prefetching *that achieves the optimal fault rate almost surely in the limit against the class of FSPs and that is simultaneously optimal in terms of running*

time for the general case of $\alpha \geq 2$ pages and cache size $k \geq 1$. ("Almost surely" means that the probability that convergence does not occur for an arbitrary sequence converges to 0 as the sequence length n gets larger.)

Our analysis model and main results are summarized in the next section. In section 3, we present our core prefetching algorithm P_1, which makes use of sampling without replacement, and we analyze it in section 4 by comparing it with the best one-state prefetcher. In section 5 we draw on ideas from information theory [9] applied to predicting [8, 12, 28] and generalize P_1 to get a universal prefetcher P that is optimal in the limit against a general FSP. The resulting optimal prefetcher P is a blend of P_1 and the prefetcher [36] based on the Lempel–Ziv data compressor [18, 25, 37]. We show in section 6 how to implement the prefetcher in constant expected time per prefetched page, independent of alphabet size α and cache size k, by using the optimal dynamic algorithm for generating discrete random variates of Matias, Vitter, and Ni [26], which uses a table lookup method of Hagerup, Mehlhorn, and Munro [16]. Other issues are discussed in section 7.

2. Analysis model and main results. We denote the cache size by k and the total number of different pages (or alphabet size) by α. We use the notation σ_i^j to denote the subsequence of a (possibly infinite) sequence σ starting at the ith page request up to and including the jth page request; in particular, σ_1^n denotes the first n page requests of σ. Given a parsing of σ_1^n into subsequences, we will denote the jth subsequence by σ_j.

DEFINITION 1. *An FSP is represented as a quintuple (S, A, g, D, z_0), where S is a finite set of states, $A = \{0, 1, 2, \ldots, \alpha - 1\}$ is a finite alphabet of cardinality $|A| = \alpha$, g is a deterministic "next state" function that maps $S \times A$ into S, D is a (possibly randomized) decision strategy function that maps S into a k-tuple A^k, and $z_0 \in S$ is the start state. The FSP prefetches at state $z \in S$ the k pages specified by $D(z)$, and upon seeing the next page request i, it changes state from z to $g(z, i)$. We denote the set of all FSPs with at most s states by $\mathcal{F}(s)$.*

We next define the best fault rate achieved on a sequence by the class of FSPs.

DEFINITION 2. *Given an FSP F and a sequence σ_1^n, we denote by $Fault_F(\sigma_1^n)$ the fault rate of F on σ_1^n, that is, the number of page faults of F on σ_1^n (expected number of page faults if F has a randomized decision strategy), divided by the length n of the sequence. We define $Fault_{\mathcal{F}(s)}(\sigma_1^n)$ to be $\inf_{F \in \mathcal{F}(s)} Fault_F(\sigma_1^n)$. With a little abuse of notation we also denote by $Fault_B(\sigma_1^n)$ the fault rate of a nonfinite state prefetcher B.*

Intuitively, we can think of $Fault_{\mathcal{F}(s)}(\sigma_1^n)$ as being given by an optimal *offline* algorithm restricted by the finite state requirement. This means that unlike an offline algorithm, the FSP does not know the sequence σ_1^n beforehand. However, it knows how many times each of its transitions will be traversed when it is used to prefetch on the sequence σ_1^n. In other words, the optimal FSP is a "weak" offline algorithm. For example, the optimal one-state FSP for a sequence σ_1^n does not know σ_1^n but knows how many times each page appears in σ_1^n. By simple convexity arguments it can be verified that the optimal one-state FSP for σ_1^n will, when at state z, deterministically prefetch the k pages corresponding to the k transitions out of z that are traversed the maximum number of times. (Hence $Fault_{\mathcal{F}(s)}(\sigma_1^n) = \min_{F \in \mathcal{F}(s)} Fault_F(\sigma_1^n)$, the minimum fault rate achieved by any FSP with at most s states on σ_1^n.) For example, $Fault_{\mathcal{F}(1)}(\sigma_1^n)$ is attained by the following one-state (zero-order) prefetcher F_1: count the number of times that page i, for $0 \leq i \leq \alpha - 1$, appears in σ_1^n. Let $C_1(\sigma_1^n) = \{i_1, i_2, \ldots, i_k\}$ be k pages with the maximum k counts. At every time t, for $1 \leq t \leq n$,

predict the next page to be one of the k pages in $C_1(\sigma_1^n)$ (that is, we always keep the same pages in cache). We design an online randomized prefetcher P_1 that achieves on average the best single-state (zero-order) prefetching fault rate $Fault_{\mathcal{F}(1)}(\sigma_1^n)$ on every sequence σ_1^n of length n in the limit as $n \to \infty$.

THEOREM 1. *For every sequence σ_1^n of length n drawn from alphabet A, the fault rate of prefetcher P_1 on σ_1^n converges almost surely to $Fault_{\mathcal{F}(1)}(\sigma_1^n)$ as $n \to \infty$. In particular,*

$$Fault_{P_1}(\sigma_1^n) \leq Fault_{\mathcal{F}(1)}(\sigma_1^n) + O\left(\frac{\log n}{\sqrt{n}}\right). \tag{1}$$

The main difficulty in developing P_1 and its proof of optimality is that the alphabet size α and the cache size k are arbitrary. We note that even for the $\alpha = 2$, $k = 1$ case, the convergence rate cannot be faster than $O(1/\sqrt{n})$ [7].

The importance of the above theorem lies in its generalization to higher order using techniques from information theory [9]. The approach of [12] allows us to combine P_1 with a prefetcher [36] based on the Lempel–Ziv data compressor [18, 25, 37] to get a prefetcher P that is optimal in the limit against the class of FSPs.

THEOREM 2. *For every sequence σ_1^n of length n drawn from alphabet A, and any $s \geq 0$, the fault rate of prefetcher P on σ_1^n converges almost surely to $Fault_{\mathcal{F}(s)}(\sigma_1^n)$ as $n \to \infty$.*

From the observation in section 1.1 that under the model from [36] (where the sequences of page requests are generated by a finite state Markov source), the optimal prefetcher is also an FSP, we get the following corollary.

COROLLARY 1. *Under the model from* [36]*, where the sequences of page requests are generated by a finite state Markov source M, the fault rate of prefetcher P converges almost surely to the minimum fault rate of any online prefetcher with complete a priori knowledge of the source M.*

The expected running time for prefetcher P can be made optimal by using the optimal dynamic random variate generator of [26].

THEOREM 3. *The prefetcher P runs in constant expected time (independent of α and k) for each page prefetched; that is, it requires an average of $O(k)$ time to determine which k pages to prefetch.*

The rate of convergence of Theorems 1 and 2 depends on the alphabet size α. For example, the error term is $O(\alpha k^2(\log n)/\sqrt{n})$ in Theorem 1; for simplicity we suppress the αk^2 term in our discussion since it is insignificant w.r.t. n in the limit. (Note that in general $k \ll \alpha$.) However, the constant time bound for each prediction is entirely independent of α and k, which is important from a computational point of view.

3. The prefetching algorithm P_1. In this section we give the algorithm P_1 that matches the best one-state prefetcher in the limit. Before introducing P_1, we present the more intuitive algorithm P_1' upon which algorithm P_1 is based.

Let t be the current time and let σ_1^t be the sequence of t pages requested until now. Let $f_i(\sigma_1^t)$, $0 \leq i \leq \alpha - 1$, denote the number of times page i appears in σ_1^t. Define $r_t = 2^j$ when $4^{j-1} < t \leq 4^j$. That is, the integer r_t is "close to" $\sqrt{t}$; it doubles at discrete time steps (when t is one greater than a power of 4).

The key idea that prefetcher P_1' (and prefetcher P_1) uses is to reduce the problem of prediction to the problem of generating random variates. Intuitively P_1' should choose for the cache the page i with the highest or nearly highest frequency

count $f_i(\sigma_1^t)$. (Randomness in the picking is required, by the remark in section 1 [7], so it does not suffice to simply pick the page with the highest frequency count; see Appendix A.1. Further, the "natural" randomized algorithm that prefetches page i at time t with probability proportional to $f_i(\sigma_1^t)$ is also not optimal as shown in Appendix A.2.) In P_1' we get the effect of choosing the pages with the highest or nearly highest counts by "boosting" the frequency counts of the page by a large power and then choosing a page with probability proportional to its boosted count. (The boosted counts will be very large but can be represented with $O(\log n)$ bits, using the scheme discussed in section 6.) Efficient random variate generation with dynamically changing weights can be done using [26], as discussed in section 6.

The algorithm P_1' is a simple randomized weighting algorithm that makes k predictions at each time step for the next page request. At each time t and $0 \le i \le \alpha - 1$, P_1' assigns to page i a probability $p_{i,t}$ proportional to the boosted frequency count

$$\left(f_i(\sigma_1^t)\right)^{r_t}, \tag{2}$$

which is the r_tth power of frequency $f_i(\sigma_1^t)$. It predicts k items for the next request by choosing without replacement from the distribution $p_{0,t}, p_{1,t}, \ldots, p_{\alpha-1,t}$.

Example 1. *Let* $\alpha = 3$, $k = 2$. *If the sequence* σ_1^t *of* $t = 9$ *pages seen until now is* 210011102, *we have* $f_0(\sigma_1^9) = 3$, $f_1(\sigma_1^9) = 4$, $f_2(\sigma_1^9) = 2$, *and* $r_t = 2^2 = 4$. *Algorithm* P_1' *assigns probability* $p_{0,9} = 3^4/(3^4 + 4^4 + 2^4) \approx 0.229$ *to page* 0, *probability* $p_{1,9} = 4^4/(3^4 + 4^4 + 2^4) \approx 0.725$ *to page* 1, *and probability* $p_{2,9} = 2^4/(3^4 + 4^4 + 2^4) \approx 0.045$ *to page* 2. *It predicts two pages out of these three by choosing without replacement based on the above probability distribution. Pages* 0 *and* 1 *are the likely pages to be chosen.* □

Algorithm P_1' can be shown to be optimal against the best one-state FSP for general $\alpha \ge 2$ but only when $k = 1$ [23]. We can modify algorithm P_1' to get algorithm P_1 that is optimal against the best one-state FSP for general $\alpha \ge 2, k \ge 1$.

DEFINITION 3. *We define subsequence* $\sigma_0 = \sigma_1^2$, *and* $\sigma_j = \sigma_{4^{j-1}+2}^{4^j+1}$ *for* $j \ge 1$. *We call* σ_j *the* jth r-subsequence *of* σ_1^n.

Notice from the definition of r_t that P_1' predicts each page in an r-subsequence using the same value for r_t in (2) when $t > 2$.

Algorithm P_1 works like algorithm P_1', except that the frequency counts for the pages are reset to 0 at the start of each r-subsequence. That is, at time $t > 1$, $4^{j-1} < t \le 4^j$, algorithm P_1 assigns to page i a probability $p_{i,t}$ proportional to

$$\left(f_i(\sigma_{4^{j-1}+2}^t)\right)^{r_t}.$$

Example 2. *As in Example* 1, *let* $\alpha = 3$, $k = 2$, *and let the sequence of* $t = 9$ *pages* σ_1^t *seen until now be* 210011102. *We have* $\sigma_0 = 21$, $\sigma_1 = 001$, *and the portion of* σ_2 *seen until now is* 1102. *The counts of pages* 0, 1, *and* 2 *in the current* r*-subsequence are* 1, 2, *and* 1, *respectively, and* $r_t = 2^2 = 4$. *Algorithm* P_1 *assigns pages* 0, 1, *and* 2 *the probabilities* $p_{0,9} = 1^4/(1^4 + 2^4 + 1^4) \approx 0.056$, $p_{1,9} = 2^4/(1^4 + 2^4 + 1^4) \approx 0.889$, *and* $p_{2,9} = 1^4/(1^4 + 2^4 + 1^4) \approx 0.056$, *respectively. It predicts two pages out of these three by choosing without replacement based on the above probability distribution.* □

This regular throwing away of past information by algorithm P_1 makes the proof of optimality more elegant. Algorithm P_1 may also perform better than algorithm P_1' in practice, since it captures the effect of locality of reference found in page request sequences [3, 5, 11, 19, 20, 33].

4. One-state case: Optimality of P_1 vs. $\mathcal{F}(1)$. In this section we prove an important special case of Theorem 1, namely, that the expected value of P_1's fault

rate $Fault_{P_1}(\sigma_1^n)$ converges to $Fault_{\mathcal{F}(1)}(\sigma_1^n)$; the almost-sure convergence follows by using the Borel–Cantelli lemma. As pointed out in section 2, given a sequence σ_1^n, the following prefetcher $F_1 \in \mathcal{F}(1)$ attains the minimum fault rate among all one-state prefetchers: count the number of times page i, for $0 \le i \le \alpha - 1$, appears in σ_1^n. Let $C_1(\sigma_1^n) = \{i_1, i_2, \ldots, i_k\}$ be the pages with the maximum k counts. For each time instant t, $1 \le t \le n$, F_1 prefetches the k pages in $C_1(\sigma_1^n)$. We have

$$Fault_{\mathcal{F}(1)}(\sigma_1^n) = 1 - \frac{f_{i_1}(\sigma_1^n) + f_{i_2}(\sigma_1^n) + \cdots + f_{i_k,n}(\sigma_1^n)}{n}. \tag{3}$$

We now define a *balanced form* of a subsequence and an *approximately balanced* form of a sequence. These notions are useful in showing the optimality of P_1.

DEFINITION 4. *A subsequence $\widehat{\sigma}_a^b$ is a* balanced form *of σ_a^b if*

1. *$\widehat{\sigma}_a^b$ has the same composition of pages as σ_a^b, that is, for all $0 \le i \le \alpha - 1$, page i appears the same number of times in $\widehat{\sigma}_a^b$ as it does in σ_a^b;*
2. *for each $a \le t \le b$, page $\widehat{\sigma}_t^t$ occurs the maximal number of times in $\widehat{\sigma}_a^t$ in comparison with the other pages.*

For example, if $\sigma_1^{10} = 1111211321$, a balanced form is $\widehat{\sigma}_1^{10} = 123121211$.

DEFINITION 5. *A sequence $\widetilde{\sigma}_1^n$ is an* approximately *or* piecewise balanced *form of σ_1^n if*

$$\widetilde{\sigma}_1^n = \widehat{\sigma}_0 \widehat{\sigma}_1 \widehat{\sigma}_2 \ldots,$$

where $\widehat{\sigma}_j$ is a balanced form of σ_j, and σ_j is the jth r-subsequence of σ_1^n as defined in Definition 3.

By (3) and the first condition in Definition 4, we have $Fault_{\mathcal{F}(1)}(\sigma_1^n) = Fault_{\mathcal{F}(1)}(\widetilde{\sigma}_1^n)$. Our strategy to show optimality of P_1 (Theorem 1) is a two-step process described by the following two theorems. First, we show that the fault rate of P_1 on σ_1^n is never more than the fault rate of P_1 on the approximately balanced $\widetilde{\sigma}_1^n$.

THEOREM 4. *For every sequence σ_1^n, we have*

$$Fault_{P_1}(\sigma_1^n) \le Fault_{P_1}(\widetilde{\sigma}_1^n). \tag{4}$$

We then compute the fault rate of P_1 on the approximately balanced $\widetilde{\sigma}_1^n$ and show that it is close to the fault rate of the best one-state machine for σ_1^n.

THEOREM 5. *For every sequence σ_1^n, we have*

$$Fault_{P_1}(\widetilde{\sigma}_1^n) - Fault_{\mathcal{F}(1)}(\sigma_1^n) = O(\log n/\sqrt{n}\,). \tag{5}$$

The proofs of the above two theorems are dealt with in the next two subsections.

4.1. The approximately balanced sequence is sufficiently worst case. In this subsection we prove Theorem 4 using an interesting extension of the switch analysis of [12] in conjunction with the important notion of boosted frequency counts (2).

We denote the jth r-subsequence σ_j by π_1^η, where $\eta = 4^j - 4^{j-1}$ is the length of σ_j. The sequence π_1^η can be converted to a balanced form $\widehat{\pi}_1^\eta$ by an iterative balancing strategy. Without loss of generality, let the $(\tau+2)$nd page request in π_1^η be 1, and let the $(\tau+1)$st page request of the balanced sequence $\widehat{\pi}_1^{\tau+1}$ be 0. We denote by f_i the number of times page i appears in $\widehat{\pi}_1^\tau$. In particular, we denote the number of 0's in $\widehat{\pi}_1^\tau$ by f_0, and the number of 1's in $\widehat{\pi}_1^\tau$ by f_1. We consider the following iterative balancing strategy to convert $\widehat{\pi}_1^{\tau+1}$ to $\widehat{\pi}_1^{\tau+2}$.

Balancing strategy. Since $\widehat{\pi}_1^{\tau+1}$ is balanced, $f_1 \leq f_0 + 1$. If $f_1 \geq f_0$, then $\widehat{\pi}_1^{\tau+1}$ appended with a 1 gives $\widehat{\pi}_1^{\tau+2}$. If $f_1 < f_0$, we perform a "01" $\rightarrow$ "10" *switch at position* $(\tau+1, \tau+2)$ by moving the 1 in front of the 0. We continue this process of "bubbling" the 1 forward through $\widehat{\pi}_1^{\tau}$ by performing similar switches, until the subsequence of the first $\tau+2$ page requests of π_1^{η} is balanced.

Our proof of Theorem 4 consists in showing that each switch in the balancing strategy does not lower the page fault rate of the entire sequence.

A similar but simpler idea worked in the binary case for a different algorithm [12], in which the sequence did not need to be broken up into subsequences, and the sequence $\widehat{\sigma}_1^n$ could be shown to be strictly worst case. We break σ_1^n into subsequences as part of our method for achieving optimal computational efficiency (as discussed in section 6).

We now show that a switch within an r-subsequence can only increase the fault rate for algorithm P_1. The fact that we allow $k \geq 1$ predictions before each page request makes the probability terms in the analysis conditional on the previous prefetches at that time step, and that complicates the analysis.

LEMMA 1. *Each switch involved in converting π_1^{η} to $\widehat{\pi}_1^{\eta}$ creates a subsequence on which P_1 has a larger fault rate (that is, switches within an r-subsequence increase the fault rate).*

Proof. Let us denote by A the probability of predicting the 0 in $\pi_1^{\tau}01\pi_{\tau+3}^{\eta}$ and by B the probability of predicting the 1 in $\pi_1^{\tau}01\pi_{\tau+3}^{\eta}$, where the 0 and 1 are in the same r-subsequence. Similarly, let us denote by C the probability of predicting the 1 in $\pi_1^{\tau}10\sigma_{\tau+3}^{\eta}$, and by D the probability of predicting the 0 in $\pi_1^{\tau}10\sigma_{\tau+3}^{\eta}$, where the 1 and 0 are in the same r-subsequence. The number of faults algorithm P_1 makes on the portions π_1^{τ} and $\pi_{\tau+3}^{\eta}$ will be the same before and after the switch, since the probability of fault by P_1 at position x of π_1^{η} depends only on the composition of pages in π_1^{x}. (Recall that P_1 throws away all previous counts for pages at the beginning of an r-subsequence.) To show that a switch increases the fault rate, we must show that the increase in the number of faults caused by moving the 0 to later in the sequence overshadows the decrease in the number of faults caused by moving the 1 to earlier in the sequence; that is, we need to show that $(1-A)+(1-B) \leq (1-C)+(1-D)$. This is equivalent to showing that

$$A - D \geq C - B. \tag{6}$$

If P_1 makes only one prediction at each step, the proof of (6) is easy. (The proof follows directly from (7) for the special case $k = 1$.) However, P_1 makes $k \geq 1$ predictions at each time instant.

Let $A = A_1 + A_2 + \cdots + A_k$, where A_i is the probability of predicting the 0 in $\pi_1^{\tau}01\pi_{\tau+3}^{\eta}$ in the ith prediction. The probabilities B, C, D are similarly partitioned. Each A_i can be further broken up into a "good part" G_i^A and a "bad part" R_i^A. The good part G_i^A is the probability of the event that we predict the 0 in $\pi_1^{\tau}01\pi_{\tau+3}^{\eta}$ in the ith prediction and that none of the previous $i-1$ predictions was a 1. The bad part R_i^A is the probability of the event that we predict the 0 in $\pi_1^{\tau}01\pi_{\tau+3}^{\eta}$ in the ith prediction and that a 1 was predicted in one of the previous $i-1$ predictions. Clearly, $A_i = G_i^A + R_i^A$, and $R_1^A = 0$. The quantities $G_i^B, R_i^B, G_i^C, R_i^C, G_i^D, R_i^D$ are defined similarly. We now show the following two facts:

Fact 1. $G_i^A - G_i^D \geq G_i^C - G_i^B$ for $1 \leq i \leq k$;

Fact 2. $(G_i^A - G_i^D) + (R_{i+1}^A - R_{i+1}^D) = 0$ and $(G_i^C - G_i^B) + (R_{i+1}^C - R_{i+1}^B) = 0$ for $1 \leq i \leq k-1$.

Facts 1 and 2 say intuitively that the good parts of the ith prediction maintain the relationship that we want. The bad parts of the ith prediction *exactly* cancel the gain from the good parts of the $(i-1)$st prediction. The lemma follows from the above two facts as follows:

$$A - D = \sum_{1 \le i \le k} A_i - D_i = \sum_{1 \le i \le k} (G_i^A - G_i^D) + (R_i^A - R_i^D) = G_k^A - G_k^D$$

by repeated application of Fact 2. Similarly, $C - B = G_k^C - G_k^B$. By Fact 1, $G_k^A - G_k^D \ge G_k^C - G_k^B$, which implies $A - D \ge C - B$.

We now prove Facts 1 and 2 by induction. Since π_1^τ is an r-subsequence, at each time step, algorithm P_1 boosts frequencies by the same exponent r_t; for convenience, we denote this exponent by r. Let $d = \sum_0^{\alpha-1} (f_i)^r$, $d_1 = d - (f_1)^r + (f_1+1)^r$, and $d_0 = d - (f_0)^r + (f_0+1)^r$. These quantities are involved in the denominators of the rational expressions for the predictions. Let $u_i^A = u_i^A(x_1, \ldots, x_{i-1})$ be the term in G_i^A that corresponds to predicting $x_1, \ldots, x_{i-1}$, none of them a 0 or a 1 as the first $i-1$ predictions and 0 as the ith prediction. (The order of the first $i-1$ predictions is important. For example, when $i = 3$, the probability of predicting a 0 following x_1, x_2 is different from the probability of predicting a 0 following x_2, x_1.) The quantities u_i^B, u_i^C, u_i^D are defined similarly. The expressions in Fact 1 can be expressed in terms of the u's; for example,

$$G_i^A - G_i^D = \sum_{\substack{x_1, \ldots, x_{i-1} \\ \text{distinct}, \neq 0, 1}} u_i^A - u_i^D.$$

Let $v_i^A = v_i^A(x_1, x_2, \ldots, x_{i-2})$ be the term in R_i^A that corresponds to predicting a 0 in the ith prediction with a 1 as one of the first $i-1$ predictions and the other $i-2$ predictions being $x_1, \ldots, x_{i-2}$, none of them a 0 or a 1. (The order of these $i-2$ predictions is important, as it is with u_i, but the relative point at which the 1 is predicted is arbitrary. In effect, v_i^A is the sum of $i-1$ probability terms, where the $i-1$ terms correspond to the $i-1$ positions that the "1" is predicted.) The quantities v_i^B, v_i^C, v_i^D are defined similarly. The expressions in Fact 2 can be expressed in terms of the u's and the v's; for example,

$$(G_i^A - G_i^D) + (R_{i+1}^A - R_{i+1}^D) = \sum_{\substack{x_1, \ldots, x_{i-1} \\ \text{distinct}, \neq 0, 1}} (u_i^A - u_i^D) + (v_{i+1}^A - v_{i+1}^D).$$

Let $den(d, 0) = d$, and let $den(d, i-1) = d - (f_{x_1})^r - \cdots - (f_{x_{i-1}})^r$ for $i \ge 2$. Let $\pi den(d, i-1)$ for $i \ge 2$ be the $(i-1)$-term falling product $d \times (d - (f_{x_1})^r) \times \cdots \times (d - (f_{x_1})^r - \cdots - (f_{x_{i-2}})^r) = \Pi_{j=1}^{i-1} den(d, j-1)$.

Proof of Fact 1. It suffices to show by induction that $u_i^A - u_i^D \ge u_i^C - u_i^B$. For the base case when $i = 1$, $u_1^A - u_1^D = (f_0)^r/d - (f_0)^r/d_1$, and $u_1^C - u_1^D = (f_1)^r/d - (f_1)^r/d_0$. Hence

$$\frac{u_1^A - u_1^D}{u_1^C - u_1^B} = \frac{(f_0)^r}{(f_1)^r} \times \frac{(f_1+1)^r - (f_1)^r}{(f_0+1)^r - (f_0)^r} \times \frac{d_0}{d_1} = \frac{(1+1/f_1)^r - 1}{(1+1/f_0)^r - 1} \times \frac{d_0}{d_1}. \tag{7}$$

Since the function $g(x) = x^r$ is convex and $f_0 \ge f_1$, the above quantity is at least 1. From the induction hypothesis that $u_{i-1}^A - u_{i-1}^D \ge u_{i-1}^C - u_{i-1}^B$ we get

$$(f_{x_1}f_{x_2}\cdots f_{x_{i-2}}f_0)^r\left(\frac{1}{\pi den(d,i-1)}-\frac{1}{\pi den(d_1,i-1)}\right)$$
$$\geq (f_{x_1}f_{x_2}\cdots f_{x_{i-2}}f_1)^r\left(\frac{1}{\pi den(d,i-1)}-\frac{1}{\pi den(d_0,i-1)}\right). \tag{8}$$

As in (7), since $g(x)=x^r$ is convex, and $f_0 \geq f_1$, we have

$$\frac{(f_0)^r}{(f_1)^r}\times\frac{d_1-d}{d_0-d}\times\frac{\pi den(d_0,i-1)}{\pi den(d_1,i-1)}\times\frac{den(d_0,i-1)}{den(d_1,i-1)}\geq 0.$$

It follows that

$$\frac{(f_{x_1}f_{x_2}\cdots f_{x_{i-1}}f_0)^r}{\pi den(d_1,i-1)}\left(\frac{1}{den(d,i-1)}-\frac{1}{den(d_1,i-1)}\right)$$
$$\geq \frac{(f_{x_1}f_{x_2}\cdots f_{x_{i-1}}f_1)^r}{\pi den(d_0,i-1)}\left(\frac{1}{den(d,i-1)}-\frac{1}{den(d_0,i-1)}\right). \tag{9}$$

Multiplying (8) by $(f_{x_{i-1}})^r/den(d,i-1)$ and adding to (9) gives us $u_i^A-u_i^D\geq u_i^C-u_i^B$.

Proof of Fact 2. To prove that $(G_i^A-G_i^D)+(R_{i+1}^A-R_{i+1}^D)=0$, it suffices to show that $(u_i^A-u_i^D)+(v_{i+1}^A-v_{i+1}^D)=0$. For the base case when $i=1$, $u_1^A-u_1^D=(f_0)^r(1/d-1/d_1)$, and

$$v_2^A-v_2^D=\frac{(f_1)^r(f_0)^r}{d(d-(f_1)^r)}-\frac{(f_1+1)^r(f_0)^r}{d_1(d_1-(f_1+1)^r)}.$$

Using the fact that $d_1=d+(f_1+1)^r-(f_1)^r$, it follows from simple algebra that $v_2^A-v_2^D+(u_1^A-u_1^D)=0$.

For the inductive step, recall that v_{i+1}^A is a sum of i probability terms, where the i terms correspond to the i positions that the "1" is predicted. (The terms are each rational expressions with different denominators.) In particular, v_{i+1}^A equals

$$\frac{(f_{x_1}f_{x_2}\cdots f_{x_{i-1}}f_0f_1)^r}{den(d,i-1)-(f_1)^r}\left(\frac{1}{\pi den(d,i)}+\frac{1}{den(d,i-2)-(f_1)^r}\left(\frac{1}{\pi den(d,i-1)}+\cdots\right)\right),$$

which can be simplified to

$$v_{i+1}^A=\frac{(f_{x_1}f_{x_2}\cdots f_0f_1)^r}{den(d,i-1)-(f_1)^r}\left(\frac{1}{\pi den(d,i)}+\frac{v_i^A}{(f_{x_1}f_{x_2}\cdots f_{x_{i-2}}f_0f_1)^r}\right).$$

Since $d_1-(f_1+1)^r=d-(f_1)^r$, we find that $v_{i+1}^A-v_{i+1}^D$ equals

$$\frac{(f_{x_1}f_{x_2}\cdots f_{x_{i-1}}f_0)^r}{den(d,i-1)-(f_1)^r}\left(\frac{(f_1)^r}{\pi den(d,i)}-\frac{(f_1+1)^r}{\pi den(d_1,i)}+\frac{v_i^A-v_i^D}{(f_{x_1}f_{x_2}\cdots f_{x_{i-2}}f_0)^r}\right). \tag{10}$$

By the induction hypothesis, $v_i^A-v_i^D=-(u_{i-1}^A-u_{i-1}^D)$. The value for $u_{i-1}^A-u_{i-1}^D$ is the expression on the left-hand side of (8). Substituting for $u_{i-1}^A-u_{i-1}^D$ in (10) we get

$$v_{i+1}^A-v_{i+1}^D=\frac{(f_{x_1}f_{x_2}\cdots f_{x_{i-1}}f_0)^r}{d-(f_{x_1})^r-\cdots-(f_{x_{i-1}})^r-(f_1)^r}\times U, \tag{11}$$

where

$$U=\frac{(f_1)^r}{\pi den(d,i)}-\frac{(f_1+1)^r}{\pi den(d_1,i)}-\frac{1}{\pi den(d,i-1)}+\frac{1}{\pi den(d_1,i-1)}.$$

The quantity $u_i^A - u_i^D$ can be expressed as

$$u_i^A - u_i^D = (f_{x_1} f_{x_2} \cdots f_{x_{i-1}} f_0)^r \left(\frac{1}{\pi den(d,i)} - \frac{1}{\pi den(d_1,i)} \right).$$

Adding the above expression to (11) and simplifying we find that $(u_i^A - u_i^D) + (v_{i+1}^A - v_{i+1}^D) = 0$. A similar analysis shows that $(G_i^C - G_i^B) + (R_{i+1}^C - R_{i+1}^B) = 0$. □

4.2. *Fault*$_{P_1}(\widetilde{\sigma}_1^n)$ is close to *Fault*$_{\mathcal{F}(1)}(\sigma_1^n)$. In this subsection we prove Theorem 5. Let $F_1 \in \mathcal{F}(1)$ be the best one-state prefetcher for σ_1^n. Let $F_1^j \in \mathcal{F}(1)$ be the best one-state prefetcher tuned for the jth r-subsequence σ_j. Let $NumFaults_B(\sigma_a^b)$ be the number of faults incurred by algorithm B on subsequence σ_a^b.

It is clear by definition that prefetcher F_1^j incurs at most as many faults on σ_j as F_1 incurs on σ_j. In other words,

$$NumFaults_{F_1}(\widehat{\sigma}_j) = NumFaults_{F_1}(\sigma_j) \geq NumFaults_{F_1^j}(\sigma_j) = NumFaults_{F_1^j}(\widehat{\sigma}_j). \tag{12}$$

Equation (12) directly implies the following lemma.

LEMMA 2. *The fault rate incurred for σ_1^n by using prefetcher F_1^j to prefetch for the jth r-subsequence σ_j, for each $j \geq 0$, is no greater than Fault$_{\mathcal{F}(1)}(\sigma_1^n)$. That is,*

$$Fault_{\mathcal{F}(1)}(\sigma_1^n) = Fault_{\mathcal{F}(1)}(\widetilde{\sigma}_1^n) = \frac{NumFaults_{F_1}(\widehat{\sigma}_1^n)}{n} \geq \frac{\sum_{j \geq 0} NumFaults_{F_1^j}(\widehat{\sigma}_j)}{n}.$$

The above lemma is useful since it is easier to compare algorithm P_1 with algorithm F_1^j on page request sequence $\widehat{\sigma}_j$ than it is to compare P_1 with F_1.

LEMMA 3. *The number of faults that algorithm P_1 incurs on $\widehat{\sigma}_j$ is close to the number of faults of the optimal one-state machine tuned for $\widehat{\sigma}_j$. In particular,*

$$NumFaults_{P_1}(\widehat{\sigma}_j) - NumFaults_{F_i^j}(\widehat{\sigma}_j) = O((\alpha k^2) j 2^j).$$

From Lemmas 2 and 3 we get

$$\begin{aligned} Fault_{P_1}(\widetilde{\sigma}_1^n) - Fault_{\mathcal{F}(1)}(\widetilde{\sigma}_1^n) &\leq \sum_j \frac{NumFaults_{P_1}(\widehat{\sigma}_j) - NumFaults_{F_1^j}(\widetilde{\sigma}_j)}{n} \\ &= O\left(\frac{\alpha k^2 \sum_j j 2^j}{n} \right) \\ &= O\left(\frac{\alpha k^2 \log n}{\sqrt{n}} \right). \end{aligned} \tag{13}$$

Theorem 5 follows from (13) and the observation following Definition 5.

We now give the proof of Lemma 3.

Proof of Lemma 3. For simplicity, we denote by π_1^η the balanced jth r-subsequence $\widehat{\sigma}_j$, where $\eta = 4^j - 4^{j-1}$ is the length of $\widehat{\sigma}_j$. Divide π_1^η into α subsequences $\pi_0, \pi_1, \ldots, \pi_{\alpha-1}$, where exactly $\alpha - i$ different pages appear in π_i. (Some of the π_i's may be empty. Notice that since π_1^η is balanced, each π_i has the same number of occurrences of each of the $\alpha - i$ pages in it.) We explicitly compute the difference between the expected number of faults of P_1 and the number of faults of F_1^j for each subsequence π_i. We need to be careful with the asymptotics involved, since the counts for the

pages are small in the earlier part of π_1^η, but $r_t = O(\sqrt{\eta})$ is relatively larger. For simplicity, we drop the subscript t from r_t in the following discussion; recall that r_t does not change in an r-subsequence.

Let $|\pi_i|$ be the length of subsequence π_i. Algorithm F_1^j incurs $|\pi_i| \times \max\{0, 1 - k/(\alpha - i)\}$ faults on π_i. Define L_i as

$$L_i = \begin{cases} \dfrac{|\pi_0|}{\alpha} + \dfrac{|\pi_1|}{\alpha - 1} + \cdots + \dfrac{|\pi_{i-1}|}{\alpha - (i-1)} & \text{if } i \geq 1, \\ 0 & \text{if } i = 0. \end{cases}$$

The quantity L_i tracks the number of times a page that appears in π_i has already appeared in $\pi_0, \pi_1, \ldots, \pi_{i-1}$. The expected number of faults incurred by algorithm P_1 on π_i is

$$\mathit{NumFaults}_{P_1}(\pi_i) = |\pi_i| - \sum_{u=1}^{|\pi_i|/(\alpha-i)} \sum_{v=0}^{\alpha-i-1} \sum_{k_1=1}^{k} \Pr(u, v, k_1), \tag{14}$$

where $\Pr(u, v, k_1)$ is the probability of predicting the $((u-1) \times (\alpha - i) + v + 1)$st page request of π_i in the k_1th prediction. For notational simplicity, let $t = (u-1) \times (\alpha - i) + v + 1$. Clearly, the tth page has appeared $L_i + u - 1$ times in π_1^{t-1}, and no page in π_i has appeared more than $L_i + u$ times in π_1^{t-1}. Since we require an upper bound on $\mathit{NumFaults}_{P_1}(\pi_i)$, we determine a lower bound for $\Pr(u, v, k_1)$ as follows. We only consider $k_1 \leq \alpha - i$, and we only consider that subset of events when pages with count greater than or equal to that of the tth page (i.e., pages appearing in π_i) were predicted in the previous $k_1 - 1$ predictions. Hence

$$\Pr(u, v, k_1) \geq (k_1 - 1)! \begin{pmatrix} \alpha - i - 1 \\ k_1 - 1 \end{pmatrix} \prod_{w=0}^{k_1-1} \frac{(L_i + u - 1)^r}{(\alpha - i - w)(L_i + u)^r + \sum_{q=0}^{i-1} (L_{q+1})^r}.$$

Notice that multiplying by $(k_1 - 1)!$ in the above equation is essential, since the order of the first $(k_1 - 1)$ predictions is significant. We get

$$\Pr(u, v, k_1) \geq (\alpha - i - 1)^{\underline{k_1 - 1}} \prod_{w=0}^{k_1-1} \frac{(L_i + u - 1)^r}{(\alpha - i - w)(L_i + u)^r + \sum_{q=0}^{i-1} (L_{q+1})^r}, \tag{15}$$

where the expression $x^{\underline{y}}$ stands for the "falling power" $x \times (x-1) \times \cdots \times (x - y + 1)$.

In our following analysis, the important intuition is that asymptotically $L_i + u - 1 \approx L_i + u$, but $(L_i)^r \ll (L_i + u)^r$. Replacing $\sum_{q=0}^{i-1} (L_{q+1})^r$ by $i(L_i)^r$, we get

$$\Pr(u, v, k_1) \geq (\alpha - i - 1)^{\underline{k_1 - 1}} \prod_{w=0}^{k_1-1} \frac{(L_i + u - 1)^r}{(\alpha - i - w)(L_i + u)^r + i(L_i)^r}. \tag{16}$$

Adding and subtracting $(L_i + u)^r$ to the numerator in (16) and simplifying, we get

$$\Pr(u, v, k_1) \geq (\alpha - i - 1)^{\underline{k_1 - 1}} \left(\prod_{w=0}^{k_1-1} \frac{(L_i + u)^r}{(\alpha - i - w)(L_i + u)^r + i(L_i)^r} - \frac{\delta_1(u, i)}{\alpha - i - w} \right), \tag{17}$$

where $\delta_1(u, i) = ((L_i + u)^r - (L_i + u - 1)^r)/(L_i + u)^r$. Adding and subtracting $(i/\alpha - i - w) \times (L_i)^r$ to the numerator of the first rational term in (17) and simplifying, we get

$$\Pr(u, v, k_1) \geq \frac{(\alpha - i - 1)^{\underline{k_1 - 1}}}{(\alpha - i)^{\underline{k_1}}} \left(1 - (\delta_1(u, i) + \delta_2(u, i))\right)^{k_1}, \tag{18}$$

where $\delta_2(u,i) = i(L_i)^r/(L_i+u)^r$. With the expression for $\Pr(u,v,k_1)$ from (18) it is easy to verify that the leading term of $\mathit{NumFaults}_{P_1}(\pi_i)$ is $|\pi_i| \times \max\{0, 1 - k/(\alpha - i)\}$, which is the number of faults incurred by F_1^j on π_i. The error term $\epsilon = \mathit{NumFaults}_{P_1}(\pi_i) - \mathit{NumFaults}_{F_1^j}(\pi_i)$ equals

$$\frac{1}{\alpha - i} \sum_{u=1}^{|\pi_i|/(\alpha-i)} \sum_{v=0}^{\alpha-i-1} \sum_{k_1=1}^{k} \epsilon(u,i,k_1), \tag{19}$$

where

$$\epsilon(u,i,k_1) = 1 - (1 - \delta_1(u,i) - \delta_2(u,i))^{k_1}.$$

The following facts can be verified by using the asymptotic techniques from [15, Chapter 9]:

1. $\delta_1(u,i) \le r/(L_i + u - 1)$ if $L_i + u - 1 \ge r$;
2. $\delta_2(u,i) \le i \times \exp(-ru/2L_i)$ if $u \le L_i$, and $\delta_2(u,i) \le i2^{-r}$ if $u \ge L_i$.

The lower-order terms arising from the binomial expansion of $(1-(\delta_1(u,i)+\delta_2(u,i))^{k_1}$ from (19) (i.e., terms of degree 2 or greater) can be disregarded if $L_i + u - 1 \ge kr$ and $u \ge \sqrt{3\eta}\log(\alpha k)$. When $L_i + u - 1 < kr$ or $u < \sqrt{3\eta}\log(\alpha k)$, we can bound $\epsilon(u,i,k_1)$ by 1; the net contribution of this to ϵ is $O(\alpha r \ln(\alpha k))$. Disregarding lower-order terms in (19) and using the expressions from Facts 1 and 2 above, we get $\epsilon = O(\alpha k^2 r \ln \eta + \alpha k^2 \eta/r) = O(\alpha k^2 \sqrt{\eta} \log \eta) = O(\alpha k^2 j 2^j)$. □

5. Generalizing P_1 to get P. In this section we prove Theorem 2 by constructing our optimal prefetcher P. The prefetcher P is a mix of P_1 and the character-based version of the Lempel–Ziv algorithm for data compression. The original Lempel–Ziv algorithm is a word-based data compression algorithm that parses the input string x_1^n into distinct substrings x_0, x_1, x_2, …, x_c such that, for all $j \ge 1$, substring x_j without its last character is equal to some x_i for $0 \le i < j$. (We use the convention that $x_0 = \lambda$, the empty substring.) It encodes the string one substring at a time. Since the substrings are prefix-closed, they can be represented by a dynamically growing tree T (the "LZ tree"), with the nodes of the tree representing the substrings and node x_i being an ancestor of node x_j if substring x_i is a prefix of substring x_j; λ is the root of the tree. An example of the LZ tree is given in Figure 1a.

Let $x(z)$ be the sequence of pages seen until now by P when at state z. At the end of a parse, prefetcher P positions itself at the root of the LZ tree. It looks at the subsequence $x(z)$ at its current state z and simulates P_1 on $x(z)$ to prefetch for the next page. (Algorithm P_1 breaks $x(z)$ into r-subsequences and prefetches based on the current r-subsequence at state z as described in section 3. Note that P_1 does not have to maintain $x(z)$ explicitly; it only has to maintain counts for the different pages.) On observing the next page request j, it updates $x(z)$, moves down the transition labeled by j, and prefetches the next page similarly by simulating P_1 on the sequence of pages seen at the new current state. On reaching a leaf state, it prefetches k pages at random, and the next request ends a parse. The important point is that although the counts for some or all of the transitions can be 0 (since algorithm P_1 resets the counts for all pages to 0 at the beginning of an r-subsequence), the transitions themselves are retained in the tree. An example snapshot of the data structure of P is given in Figure 1b.

We now briefly explain why P is optimal against an arbitrary s-state machine (Theorem 2) using the interesting approach of [12]. An mth-order Markov prefetcher

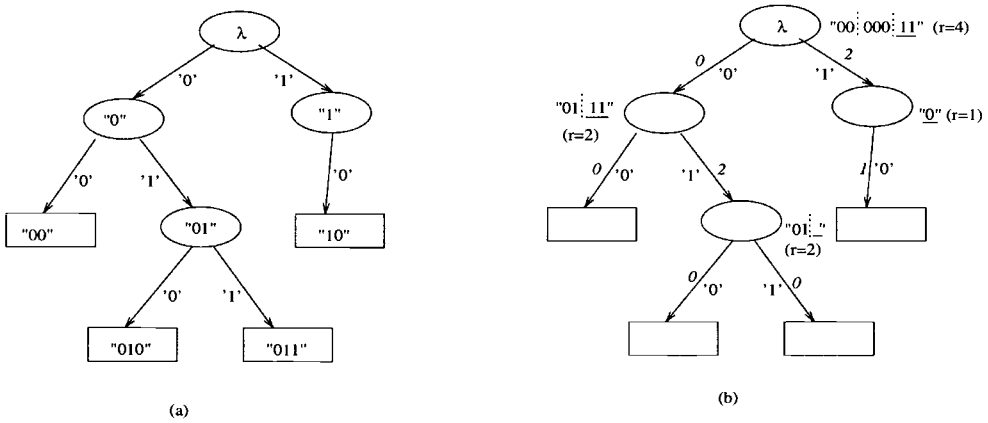


FIG. 1. *Snapshot of data structure for algorithm P. Assume for simplicity that our alphabet is* $\{0,1\}$. *We consider the page request sequence* x_1^t = "00001010011110 ...." *The Ziv–Lempel encoder parses this string as "*(0)(00)(01)(010)(011)(1)(10) ...." *The tree T that is built at the end of the seventh parse is pictured above in* (a). *In* (b), *next to each node/state z of the tree we give the sequence of page requests* $x(z)$ *seen at that state. For example, for any page request sequence* x_1^t *that is parsed by the Lempel–Ziv data compressor into distinct substrings* λ, x_1, x_2, ..., x_c, *the first page of each substring* x_i, $1 \le i \le c$, *forms* $x(\lambda)$, *the sequence of pages requested when the current state is the root of the tree. In* (b), $x(\lambda) = 0000011$. *The dotted vertical lines in the sequences delimit the r-subsequences, and the underlined portion is the current r-subsequence. The counts (given in italics) on the transitions out of each state z are the counts obtained by simulating* P_1 *on* $x(z)$.

predicts its k choices for the next page based solely on the previous m page requests of the sequence. In particular, an mth-order prefetcher can be described by an FSP having α^m states, where each state is labeled by an m-page context (denoting the last m pages requested), and the transitions denote the unique change from one m-context to the next. The pages to prefetch are determined solely by the state that the mth-order Markov prefetcher is in, which is equivalent to the most recent m pages requested. The basic idea of the proof is to compare both prefetcher P and the best s-state prefetcher with mth-order Markov prefetchers.

If we let m be large, an mth-order Markov prefetcher achieves, for every sequence σ_1^n and any s, a fault rate close to the fault rate of the best s-state prefetcher. In particular, by simple extensions to [12, Theorem 2] as shown in Appendix B, we see that

$$Fault_{M(\alpha^m)}(\sigma_1^n) \le Fault_{\mathcal{F}(s)}(\sigma_1^n) + O\left(\sqrt{\frac{\log s}{m+1}}\right). \tag{20}$$

The idea of the proof in [12] is to consider a "cross-product" machine of the mth-order Markov predictor and the s state prefetcher and to show that the cross-product machine is not much better than either of its constituents.

The prefetcher P can be looked upon as a Markov prefetcher of growing order. In particular, most nodes in the tree T built by P (i.e., nodes below a depth of m) have a context of length greater than m. Hence one would intuitively expect that in the limit as $n \to \infty$, prefetcher P will "beat" any mth-order Markov predictor. Theorem 1 can be applied to each node of T, and by carefully summing the errors

over each node we get

$$Fault_P(\sigma_1^n) \leq Fault_{M(\alpha^m)}(\sigma_1^n) + \delta(n,m), \tag{21}$$

where for a fixed m, $\delta(n,m) = O((\log\log n)/\sqrt{\log n})$. This idea was used in [12, Theorem 4] for a binary alphabet, and the simple changes required to obtain (21) are summarized in Appendix B. Theorem 2 follows from (20) and (21).

Given that P_1 is optimal against $\mathcal{F}(1)$ (Theorem 1), in order to show optimality of P against $\mathcal{F}(s)$ by the approach described above, we need to extend some results of [12] to hold for the prefetching problem. These extensions are simple as described in Appendix B. The intuition for why these extensions are simple is because the comparisons are primarily between two "offline" algorithms, and the online algorithm is not much involved, as opposed to the more complex analysis of section 4.

6. Constant-time prediction. In this section we prove Theorem 3 by showing that our prefetcher P runs in constant time (independent of α, k) on the average for each of the pages it prefetches into cache.

In section 5, we showed that it suffices to consider one-state prefetchers; the prefetcher at each step uses the appropriate P_1 to generate random variates according to a dynamically changing set of weights. We showed earlier that P_1's prediction strategy is optimal, in which we successively pick a page at random (without replacement) with probabilities in proportion to the boosted frequency counts $(f_0)^r$, $(f_1)^r$, $\ldots$, $(f_{\alpha-1})^r$, where $r \approx \sqrt{t}$. (Actually, we use $r = 2^j$, where $4^{j-1} < t \leq 4^j$, so that r seldom changes. The frequency counts f_i are reset to zero when r changes.)

The general problem of generating a random variate with a value in the range $\{0, 1, 2, \ldots, \alpha - 1\}$ and distributed according to α dynamically changing weights $w_0, w_1, w_2, \ldots, w_{\alpha-1}$ is solved optimally by Matias, Vitter, and Ni [26, section 5]. The idea at an intuitive level is to group the weights into ranges according to their values. Range j stores weights in the range $[2^j, 2^{j+1})$. Each range is said to have a weight equal to the sum of the weights it contains. With high probability, the individual weight chosen during the generation will be within the first $O(\log\alpha)$ ranges, so each successive group of $O(\log\alpha)$ ranges should be processed in a recursive data structure according to the weights of the ranges. The use of the rejection method [21] is used to adjust the probabilities of generation appropriately, since the weights in each bucket may vary by a factor of 2. After two recursive levels, the problem reduces to generating one of $O(\log\log\alpha)$ weights, each in the range $[1, \log\alpha]$, which can be done dynamically in constant time by the clever table lookup method of Hagerup, Mehlhorn, and Munro [16].

There is also extensive concern in [26] about the choice of hashing parameters in the universal hashing schemes used to get linear space, since no a priori bound on the key values is known. (In fact, a constant-time solution to the general dictionary problem is proposed in [26].) The model of computation allows arithmetic computation and truncated logarithms of quantities up to value $O(W)$, where W is the maximum weight.

In our application, the computation assumption of [26] is unreasonable, since it allows constant-time operations on arbitrary numbers of bits. We make the stronger requirement often used in algorithm design that the standard arithmetic operations (such as addition, multiplication, division, and using exponents and logarithms) take constant time with finite-precision quantities of $O(\log n)$ bits, where n is the length of the sequence of page requests. However, the boosted frequencies $w_i = (f_i)^r$ used in the random variate generation can be as large as $n^{\sqrt{n}}$ in value, which cannot be

manipulated efficiently. Fortunately we can get around the precision problem by approximating w_i by $2^{\lceil r \lg f_i \rceil}$. The first level of the algorithm in [26] is applied to these approximated weights, using arithmetic on the exponents, which involves only $O(\log n)$ bits. For example, we can determine the bucket j that contains $2^{\lceil r \lg f_i \rceil}$ in constant time using operations on $O(\log n)$ bits by noting that j can be represented with only about $\lg\lg((f_i)^r) = \lg r + \lg\lg f_i \leq 2 \lg n$ bits. The range j can be computed, therefore, in constant time using $O(\log n)$-bit arithmetic. The resulting recursive subproblems have polynomial-sized weights, and the rest of the construction continues as in [26].

Because of the initial approximation, if the "approximated" page i is selected for generation, a final acceptance–rejection test must be done before actually choosing page i; the acceptance probability is $(f_i)^r / 2^{\lceil r \lg f_i \rceil}$, which is at least $1/2$. This test can be done conceptually by generating a uniform random integer U in the range $[1, 2^{j+1})$ and testing if $U \leq (f_i)^r$, but handling quantities of that magnitude is infeasible, as mentioned above. It suffices to determine if $\lg U \leq r \lg f_i$, which can be done in constant time using finite precision by generating the exponentially distributed random variate $\lg U$ directly [21, p. 128]. The expected number of steps needed before the acceptance or rejection is determined is a small constant, so finite precision suffices. This completes the proof of Theorem 3.

7. Conclusions. We have studied the problem of prediction of sequences (of pages requests, for example) drawn from a finite but arbitrary alphabet of cardinality α, in which we can make, at each time step, k predictions for the next item (page). This corresponds to the problem of pure prefetching in databases. We have developed a simple randomized weighting algorithm P_1 and have combined it with the Lempel–Ziv data compressor to get an efficient prefetcher P. We have shown analytically that P's fault rate converges almost surely to that of the best FSP for every (worst-case) sequence of page requests. It has been shown in [7] that any optimal algorithm for the binary alphabet case has to be necessarily randomized. Because of the way our algorithm is designed, we need to spend at most constant expected time making the random choices for each prediction, which is optimal. Thus the algorithm is simultaneously optimal with respect to fault rate and running time.

An open problem is to study if there are stronger analysis models closer to the competitive model that would permit prediction problems such as prefetching to be studied. It would also be interesting to improve the convergence bounds while maintaining optimal running time.

We have also investigated nonpure prefetching in [10, 35], in which there may not always be enough time to load the cache with k pages before the user issues the next page request, and prefetching requests may have to be done in advance. We have also described therein a nice way of gathering the statistics with no I/O overhead. The resulting prefetcher is practical in terms of time, disk accesses, and fault rate and outperforms other known prefetchers. We also expect that our results apply to prefetchers based on data compression methods other than Lempel–Ziv that are optimal in various models.

Appendix A. Nonoptimal prefetchers. In section A.1 we give a simple example which illustrates that no deterministic algorithm can be optimal for prefetching in the worst case. In section A.2, we show that the *Proportional* algorithm, which at time t prefetches page i with probability proportional to $f_i(\sigma_1^t)$, where $f_i(\sigma_1^t)$, for $0 \leq i \leq \alpha - 1$, denotes the number of times page i appears in σ_1^t, is not optimal.

For both proofs of nonoptimality, we develop a page request sequence with alphabet $A = \{0, 1\}$, cache size $k = 1$, and compare it with the best 1-state prefetcher.

From an intuitive standpoint, algorithm *Proportional* is too conservative, while deterministic algorithms are "too naive" and hence susceptible to worst-case-type adversaries (like FSPs). Algorithm P_1, by choosing the high probability pages with very high likelihood, closely tracks the optimal. The tricky issues are proving the optimality of P_1 and making predictions in a computationally efficient fashion.

A.1. Deterministic algorithms. It is easy to see that the fault rate of an optimal 1-state prefetcher for any sequence with $\alpha = 2$ and $k = 1$ is at most $1/2$, since it predicts at each time instant the overall most frequent page. We can make a deterministic algorithm fault at every page request by creating a sequence such that the next page request is for the page not in cache.

A.2. Algorithm *Proportional***.** Consider the page request sequence

$$\sigma_1^n = 01010101 \ldots 010000 \ldots 00,$$

where the first "0101..." subsequence is of length $n/2$, and it is followed by $n/2$ 0's. The fault rate of the optimal 1-state prefetcher (which always predicts "0" to be the next page request) is $1/4$. Algorithm *Proportional* is expected to incur at least $1/2 \times n/2$ faults in the first $n/2$ page requests. Since the proportion of 0's in the entire sequence is $3/4$, algorithm *Proportional* incurs, on the average, more than $1/4 \times n/2$ faults for the last $n/2$ page requests. This implies a net average fault rate of more than $3/8$, which is clearly suboptimal.

Appendix B. Proof of Theorem 2. In this section we continue the discussion from section 5 and present the required extensions to the results of [12] in order to prove optimality of prefetcher P. (Recall that [12] deals with the binary alphabet case, which corresponds to $\alpha = 2$, $k = 1$.) The main objective of this section is to show the necessary extensions required to prove (20) and (21) (which are the counterparts for Theorems 2 and 4 from [12]).

We start with a definition of mth-order Markov prefetchers.

DEFINITION 6. *An* mth-order Markov prefetcher *prefetches for its next page based solely on the previous* m *page requests of the sequence. Using the notation from Definition* 1*, an* m*th-order Markov prefetcher has* α^m *states, where each state is (represents) an* m*-context* $(x_1, x_2, \ldots, x_m)$*, and* $g((x_1, \ldots, x_m), u) = (x_2, \ldots, x_m, u)$*. We denote the fault rate of an* m*th-order prefetcher by* $Fault_{M(\alpha^m)}(\sigma_1^n)$*, where* $M(\alpha^m) \subseteq \mathcal{F}(\alpha^m)$*.*

We also introduce notation to help describe faults easily.

DEFINITION 7. *Given an* α*-probability vector* $\vec{p} = (p_0, \ldots, p_{\alpha-1})$*, we denote by* $\min_{\alpha-k}(\vec{p})$ *the sum of the minimum* $\alpha - k$ *elements of* $\vec{p}$*. In other words if* $p_0 \geq p_1 \geq \cdots \geq p_{\alpha-1}$*, then* $\min_{\alpha-k}(\vec{p}) = \sum_{i=k}^{\alpha-1} p_i$*.*

Proving (20) is equivalent to showing that Theorem 2 from [12] holds for prefetching. As mentioned in section 5, the proof of [12, Theorem 2] is based on comparing the s-state prefetcher and the mth-order Markov prefetcher with a cross product of these two machines.[1] Theorem 2 of [12] is strongly dependent on [12, Lemma 1], where the mth-order Markov machine is compared with the cross-product machine. The basic idea of [12, Lemma 1] is to bound prediction error by a function of the

[1] The term "cross product" is taken from [36], where a similar product of two machines was used to prove the optimality of prefetchers under a Markov source model.

empirical entropies, and it is based on the following fact: for every $0 \leq p, q \leq 1$,

$$p \log \frac{p}{q} + (1-p) \log \frac{1-p}{1-q} \geq \frac{2}{\ln 2} \left(\min\{p, 1-p\} - \min\{q, 1-q\}\right)^2. \tag{22}$$

It is not hard to see that the left-hand side of (22) is closely related to the entropy and the right-hand side of (22) to the prediction error. (The idea of bounding error for prefetching by bounding coding length differences was used independently in [36] to derive optimal prefetchers from data compressors under a Markov source input model.) A fact equivalent to (22) that we prove for our prefetching problem (via Lemmas 4 and 5 below) is the following: given two probability vectors $\vec{p}$ and $\vec{q}$,

$$\sum_{1}^{\alpha} p_i \ln \frac{p_i}{q_i} \geq \frac{1}{2} \left(\min_{\alpha-k}(\vec{p}) - \min_{\alpha-k}(\vec{q}) \right)^2. \tag{23}$$

The proof of [12, Lemma 1] follows from (22), Jensen's inequality, and the convexity of the square function. The proof of [12, Theorem 2] follows from [12, Lemma 1], Jensen's inequality, the concavity of the square root function, and the chain rule of conditional entropies. Except for (22), the other aspects of the proof of [12, Theorem 2] are effectively independent of the alphabet and cache size. To derive (20), we consider the proof of [12, Theorem 2] and uniformly replace $\min\{p_0, p_1\}$, where p_0 and p_1 are the probabilities of a 0 and a 1, by $\min_{\alpha-k}(\vec{p})$, where $\vec{p}$ is the corresponding α-probability vector for prefetching, replace summations over $\{0, 1\}$ by summations over the alphabet A, and use (23) in place of (22).

We now prove (23) using the following two lemmas.

LEMMA 4. *Given two α-probability vectors $\vec{p}$ and $\vec{q}$, we have*

$$\min_{\alpha-k}(\vec{p}) - \min_{\alpha-k}(\vec{q}) \leq \sum_{i=0}^{\alpha-1} |p_i - q_i|.$$

Proof. Without loss of generality assume $p_0 \geq p_1 \geq \cdots \geq p_{\alpha-1}$. Let $X = \{0, 1, \ldots, k-1\}$, and let $Y = \{k, k+1, \ldots, \alpha-1\}$. Hence $\min_{\alpha-k}(\vec{p}) = \sum_{i \in Y} p_i$. Let $Z = \{i_1, i_2, \ldots, i_{\alpha-k}\}$ be the $\alpha - k$ pages with minimum count in $\vec{q}$. Let $U = Z \cap Y$, and $V = Z \cap X$. By definition, $\min_{\alpha-k}(\vec{p}) - \min_{\alpha-k}(\vec{q}) = \sum_{i \in Y} p_i - \sum_{i \in Z} q_i$. Since by assumption $p_0 \geq p_1 \geq \cdots \geq p_{\alpha-1}$, we have

$$\min_{\alpha-k}(\vec{p}) - \min_{\alpha-k}(\vec{q}) \leq \sum_{i \in U} (p_i - q_i) + \sum_{i \in V} (p_i - q_i) \leq \sum_{i \in A} |p_i - q_i|,$$

where A is the alphabet as given in Definition 1. □

The next lemma is well known; the proof can be found in [2, 36]. The summation on the right-hand side of the equation in the lemma is the Kullback–Leibler divergence of $\vec{q}$ w.r.t. $\vec{p}$.

LEMMA 5. *Given two probability vectors $(p_0, \ldots, p_{\alpha-1})$ and $(q_0, \ldots, q_{\alpha-1})$, we have*

$$\left(\sum_{i=0}^{\alpha} |p_i - q_i| \right)^2 \leq 2 \sum_{i=1}^{\alpha} p_i \ln \frac{p_i}{q_i}.$$

To verify (21) (i.e., the equivalent of [12, Theorem 4] for prefetching), the proof technique presented in [12] carries over with virtually no change; we need to use

Theorem 1 of our paper in place of [12, Theorem 1]. (As done earlier, we do need to uniformly replace $\min\{p_0, p_1\}$ by $\min_{\alpha-k}(\vec{p})$ and replace summations over $\{0,1\}$ by summations over the alphabet A.) The basic idea of the proof is to consider separately each node of the tree T created by P, look at the faults for the pages requested when at that node, and take the average of these individual faults weighted by the number of times the node is visited. The main observation is that for most nodes (i.e., nodes below a depth of m), there is a mapping from the nodes of T to the states of the mth-order Markov prefetcher; hence bounding the error at each node of T involves comparing with the best one-state prefetcher, which is done in Theorem 1. Since there are at most $c = O(n/\log n)$ nodes in T [37], and the one-state error from Theorem 1 is $O((\log n)\sqrt{n}/n)$, the net error turns out to be $O((\log(n/c))\sqrt{n/c}/(n/c)) = O((\log\log n)/\sqrt{\log n})$. (This is as opposed to the case of $\alpha = 2$, $k = 1$ studied in [12],where the one-state error is $O(1/\sqrt{n})$, yielding a net error of $O(1/\sqrt{\log n})$.)

This completes our description of the extensions to [12] required to prove Theorem 2.

REFERENCES

[1] D. Aldous and U. Vazirani, *A Markovian extension of Valiant's learning model*, in Proc. 31st Annual IEEE Symposium on Foundations of Computer Science, October 1990, pp. 392–396.

[2] Y. Amit and M. Miller, *Large Deviations for Coding Markov Chains and Gibbs Random Fields*, Technical Report, Washington University, St. Louis, MO, 1990.

[3] L. A. Belady, *A study of replacement algorithms for virtual storage computers*, IBM Systems J., 5 (1966), pp. 78–101.

[4] D. Blackwell, *An analog to the minimax theorem for vector payoffs*, Pacific J. Math., 6 (1956), pp. 1–8.

[5] A. Borodin, S. Irani, P. Raghavan, and B. Schieber, *Competitive paging with locality of reference*, in Proc. 23rd Annual ACM Symposium on Theory of Computation, May 1991.

[6] T. F. Chen and J. L. Baer, *Reducing memory latency via non-blocking and prefetching caches*, in Proc. 5th Internat. Conf. on Architectural Support for Programming Languages and Operating Systems, Department of Computer Science and Engineering, University of Washington, Boston, MA, October 1992.

[7] T. M. Cover, *Behavior of predictors of binary sequences*, in Proc. 4th Prague Conference on Information Theory, Statistical Decision Functions, Random Processes, Publishing House of the Czechoslovak Academy of Sciences, Prague, 1967, pp. 263–272.

[8] T. M. Cover and A. Shenhar, *Compound Bayes predictors with apparent Markov structure*, IEEE Trans. Systems Man Cybernet., V SMC-7 (1977), pp. 421–424.

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

[10] K. Curewitz, P. Krishnan, and J. S. Vitter, *Practical prefetching via data compression*, in Proc. 1993 ACM SIGMOD International Conference on Management of Data, May 1993, pp. 257–266.

[11] P. J. Denning, *Working sets past and present*, IEEE Trans. Software Engrg., SE-6 (1980), pp. 64–84.

[12] M. Feder, N. Merhav, and M. Gutman, *Universal prediction of individual sequences*, IEEE Trans. Inform. Theory, IT-38 (1992), pp. 1258–1270.

[13] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young, *On competitive algorithms for paging problems*, J. Algorithms, 12 (1991), pp. 685–699.

[14] R. G. Gallager, *Information Theory and Reliable Communication*, Wiley, New York, 1968.

[15] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics*, Addison–Wesley, Reading, MA, 1989.

[16] T. Hagerup, K. Mehlhorn, and I. Munro, *Optimal algorithms for generating time varying discrete random variables*, in Proc. of the 20th Annual International Coll. on Automata Languages and Prog., Lecture Notes in Comput. Sci. 700, Springer, New York, 1993, pp. 253–264.

[17] J. F. Hannan, *Approximation to Bayes Risk in Repeated Plays*, Contributions to the Theory of Games, Vol. 3, Annals of Mathematical Studies, Princeton University Press, Princeton, NJ, 1957, pp. 97–139.

[18] P. G. Howard and J. S. Vitter, *Analysis of arithmetic coding for data compression*, Invited paper in Special Issue on Data Compression for Images and Texts, Inform. Process. Management, 28 (1992), pp. 749–763.

[19] S. Irani, A. R. Karlin, and S. Phillips, *Strongly competitive algorithms for paging with locality of reference*, in Proc. 3rd Annual ACM-SIAM Symposium of Discrete Algorithms, January 1992.

[20] A. R. Karlin, S. J. Phillips, and P. Raghavan, *Markov paging*, in Proc. 33rd Annual IEEE Conference on Foundations of Computer Science, October 1992, pp. 208–217.

[21] D. Knuth, *The Art of Computer Programming, Vol.* 2: *Seminumerical Algorithms*, 2nd ed., Addison-Wesley, Reading, MA, 1981.

[22] D. F. Kotz and C. S. Ellis, *Prefetching in File Systems for MIMD Multiprocessors*, IEEE Transactions on Parallel and Distributed Systems, Vol. 1, April 1990, pp. 218–230.

[23] P. Krishnan and J. S. Vitter, *Optimal Prediction for Prefetching in the Worst Case*, Technical Report DUKE–CS–93–26, Duke University , Durham, NC, 1993.

[24] P. Laird, *TDAG*: *An Algorithm for Learning to Predict Discrete Sequences*, FIA-92-01, NASA Ames Research Center, AI Research Branch, Moffet Field, CA, 1992.

[25] G. G. Langdon, *A Note on the Ziv–Lempel Model for Compressing Individual Sequences*, IEEE Trans. Inform. Theory, Vol. 29, March 1983, pp. 284–287.

[26] Y. Matias, J. S. Vitter, and W. C. Ni, *Dynamic generation of discrete random variates*, in Proc. 4th Annual SIAM/ACM Symposium on Discrete Algorithms, Austin, TX, January 1993, pp. 361–370.

[27] L. A. McGeoch and D. D. Sleator, *A strongly competitive randomized paging algorithm*, Algorithmica, 6 (1991), pp. 816–825.

[28] N. Merhav and M. Feder, *Universal Sequential Learning and Decision from Individual Data Sequences*, in Proc. 5th ACM Workshop on Computational Learning Theory, Santa Cruz, July 1992.

[29] T. C. Mowry, M. S. Lam, and A. Gupta, *Design and evaluation of a compiler algorithm for prefetching*, in Proc. 5th International Conference on Architectural Support for Programming Languages and Operating Systems Computer Systems Laboratory, Boston, MA, October 1992.

[30] M. Palmer and S. Zdonik, *Fido: A cache that learns to fetch*, in Proc. 1991 International Conference on Very Large Databases, Barcelona, September 1991.

[31] A. Rogers and K. Li, *Software support for speculative loads*, in Proc. 5th International Conference on Architectural Support for Programming Languages and Operating Systems, Department of Computer Science, Boston, MA, October 1992.

[32] K. Salem, *Adaptive Prefetching for Disk Buffers*, CESDIS, Goddard Space Flight Center, Greenbelt, MD, TR–91–46, January 1991.

[33] G. S. Shedler and C. Tung, *Locality in page reference strings*, SIAM J. Comput., 1 (1972), pp. 218–241.

[34] D. D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Communications of the ACM, 28 (1985), pp. 202–208.

[35] J. S. Vitter, K. Curewitz, and P. Krishnan, *Online Background Predictors and Prefetchers*, Duke University, United States Patent No. 5,485,609, January 16, 1996.

[36] J. S. Vitter and P. Krishnan, *Optimal prefetching via data compression*, J. Assoc. Comput. Mach., 143 (1996), pp. 771–793.

[37] J. Ziv and A. Lempel, *Compression of individual sequences via variable-rate coding*, IEEE Trans. Inform. Theory, 24 (1978), pp. 530–536.

SIAM J. COMPUT.
Vol. 27, No. 6, pp. 1637–1670, December 1998
© 1998 Society for Industrial and Applied Mathematics
007

A CORRECTNESS CONDITION FOR HIGH-PERFORMANCE MULTIPROCESSORS*

HAGIT ATTIYA† AND ROY FRIEDMAN‡

Abstract. *Hybrid consistency*, a consistency condition for shared memory multiprocessors, attempts to capture the guarantees provided by contemporary high-performance architectures. It combines the expressiveness of *strong* consistency conditions (e.g., sequential consistency, linearizability) and the efficiency of *weak* consistency conditions (e.g., pipelined RAM, causal memory). Memory access operations are classified as either *strong* or *weak*. A global ordering of strong operations at different processes is guaranteed, but there is very little guarantee on the ordering of weak operations at different processes, except for what is implied by their interleaving with the strong operations. A formal and precise definition of this condition is given and an algorithm for providing hybrid consistency on distributed memory machines is presented. The response time of the algorithm is proved to be within a constant multiplicative factor of the (theoretical) optimal time bounds.

Key words. distributed shared memory, consistency conditions, sequential consistency, hybrid consistency, weak consistency

AMS subject classifications. 68-02, 68M07, 68P05, 68Q10, 68Q20, 68Q22, 68Q60, 68Q65

PII. S0097539795289215

1. Introduction.

Shared memory is an attractive paradigm for communication among computing entities because it is familiar from the uniprocessor case, it is more high level than message passing, and many of the classical solutions for synchronization problems were developed for shared memory. The fundamental problem is how to provide programmers with a useful model of logically shared data that can be accessed atomically, without sacrificing performance. The model must specify how the data can be accessed and what guarantees are provided about the results.

To enhance performance (e.g., response time), many implementations maintain multiple copies of the same logical piece of shared data (caching). Also, multiple application programs must be able to execute concurrently. More complications arise because at some level, each access to shared data has duration in time, from its start to its end; it is not instantaneous. A consistency mechanism guarantees that operations will appear to occur in some ordering that is consistent with some condition. Much research has addressed the issue of consistency for various system types and levels of abstraction. A major issue is which consistency condition should be supported: which conditions can be implemented efficiently, which conditions can be used conveniently, and which conditions support faster programs.

Until recently, theoretical research on this subject addressed *strong* consistency conditions like sequential consistency and linearizability [14, 21, 22, 24, 29, 41, 44, 45, 48, 51, 54]. These conditions guarantee that operations appear to be executed atomically, in some sequential order that is consistent with the order seen at individual

* Received by the editors July 17, 1995; accepted for publication (in revised form) August 30, 1996; published electronically June 3, 1998. This research was supported by grant 92-0233 from the United States-Israel Binational Science Foundation (BSF), Jerusalem, Israel, and Technion V.P.R.—B. and G. Greenberg Research Fund (Ottawa). An extended abstract of this work appears in *Proceedings of the* 24*th ACM Symposium on Theory of Computing,* May 1992, pp. 679–690.
http://www.siam.org/journals/sicomp/27-6/28921.html

† Department of Computer Science, Technion, Haifa 32000, Israel (hagit@cs.technion.ac.il). Part of this work was performed while the author was visiting DEC Cambridge Research Laboratory.

‡ Department of Computer Science, Cornell University, Ithaca, NY 14850 (roy@cs.cornell.edu).

processes.[1] Sequential consistency and linearizability provide a clean and easy semantics for the execution of operations; programming using these conditions is (relatively) easy.

Unfortunately, supporting either sequential consistency or linearizability has a nonnegligible cost (cf. [12, 29, 40, 46]); i.e., these consistency conditions cannot be implemented efficiently. A way around this cost is to define conditions which provide weaker guarantees on the ordering of operations and can be efficiently implemented. Prominent among these weaker conditions are *mixed* consistency conditions (e.g., [1, 20, 30, 35, 38, 51]) that distinguish between two types of operations—strong and weak. Typically, strong operations appear to be executed atomically, in some sequential order that is consistent with the order seen at individual processes.[2] The only guarantees provided for weak operations are those implied by their interleaving with strong operations. Mixed conditions can be implemented so that weak operations are extremely fast, without degrading the response time for strong operations. Thus, a programmer can use the fast weak operations most of the time and resort to the slower strong operations only when global coordination is required.

In this paper, we make a step towards a theoretical study of mixed conditions for shared memory consistency, in an attempt to further our understanding of the issues involved in selecting and implementing a memory consistency condition.

Our first contribution is a formal and precise definition of *hybrid consistency*, which is a specific mixed condition. Very informally, hybrid consistency guarantees two properties.

1. Strong operations appear to be executed in some sequential order.
2. If two operations are invoked by the same process and one of them is strong, then they appear to be executed in the order in which they were invoked.

In particular, the second property guarantees that a strong operation appears to be executed after any operation (weak or strong) invoked before it by the same process, and before any operation (weak or strong) invoked after it by the same process. The definition applies to any collection of objects for which a sequential specification is provided (cf. [41]).

Our definition is very high-level and abstract—it describes the way operations appear to the programmer, not the way they are implemented. This results in a relatively simple definition and allows us to optimize its implementation.

We believe that hybrid consistency supports common concurrent programming techniques. In section 4, we show how Peterson's simple mutual exclusion algorithm [50] can be modified to exploit hybrid consistency. We discuss the performance benefits achieved by using hybrid consistency and illustrate how the formal definition is used when arguing about the correctness of algorithms. The correctness proof of this algorithm is quite similar to the proof of Peterson's original algorithm, which assumes sequential consistency. This may indicate that the complexity of arguing about programs assuming hybrid consistency is not much greater than the complexity of arguing about them assuming sequential consistency.

Our second contribution is an algorithm that implements hybrid consistency on distributed memory machines [14, 20, 45, 51]; our algorithm supports read/write objects and assumes each processor has a copy of every object. The algorithm is completely asynchronous. Weak operations are executed instantaneously, while the

[1] These conditions are similar in flavor to the notion of *serializability* from database theory [15, 49]; however, serializability applies to *transactions* which aggregate many operations.

[2] In some cases, e.g., [1, 35, 38], the requirements from strong operations are slightly weaker.

response time for strong operations is linear in the network delay. The algorithm is based on an atomic broadcast mechanism for sending messages. This mechanism is used to guarantee that all strong operations are executed by all processes in the same order. All weak writes to the same location are executed in the same order by all processes. Combined with the fact that a strong operation does not start until all previous operations have been executed by all processes, and does not terminate until it was executed by all other processes, this guarantees the second property of hybrid consistency. By adapting proof techniques from [12, 46], we show that the response time of our algorithm is within a constant factor of the optimum. Algorithms for providing hybrid consistency for other objects, using somewhat different techniques, appear in [33].

Another class of consistency conditions are those that, essentially, include only weak operations, e.g., [5, 39, 46]. It has been shown that without strong operations, mutual exclusion can be solved only by centralized algorithms [10, 32]. We do not address these conditions any further in this paper.

The rest of the paper is organized as follows: section 2 includes our basic definitions and some notation. Section 3 contains the definition of *hybrid consistency* and a discussion of related definitions. Section 4 provides an example of programming with hybrid consistency. In section 5, we present an algorithm for providing hybrid consistency, together with its correctness proof and performance analysis. Section 6 contains lower bounds on the response time of strong operations under hybrid consistency. We conclude, in section 7, with a discussion of our results and directions for further research.

2. The system. We consider a collection of application programs running concurrently and communicating via virtual shared memory, which consists of a collection of *objects*. We assume a system consisting of a collection of nodes P connected via a communication network; each application program runs on a different node. The shared memory abstraction is implemented by a *memory consistency system* (mcs), which uses local memory and some protocol executed by the mcs processes (one at each node). A correctness condition is defined at the interface between the application programs (written by the user) and the mcs processes (supplied by the system). Thus, the mcs must provide the proper semantics when the values of the responses to calls are considered, throughout the network. An illustration of the system architecture is given in Figure 1.

The following *events* may occur at the mcs process on node i.

1. *Call events:* The initiation of operations by the application program according to their specification. For example, in the case of read/write objects, the call events are $\text{Read}_i(X)$ and $\text{Write}_i(X, v)$ for all objects X and values v.
2. *Response events:* The response of the mcs to operations initiated by the application program, according to their specification. For example, in the case of read/write objects, the response events are $\text{Return}_i(X, v)$ and $\text{Ack}_i(X)$ for all objects X and values v.
3. *Message-receive events:* $\text{receive}(i, m, j)$ for all messages m and mcs processes p_i and p_j: the mcs process on node i receives message m from the mcs process on node j.
4. *Message-send events:* $\text{send}(i, m, j)$ for all messages m and mcs processes p_i and p_j: the mcs process on node i sends message m to the mcs process on node j.

The call and message-receive events are *interrupt events*.

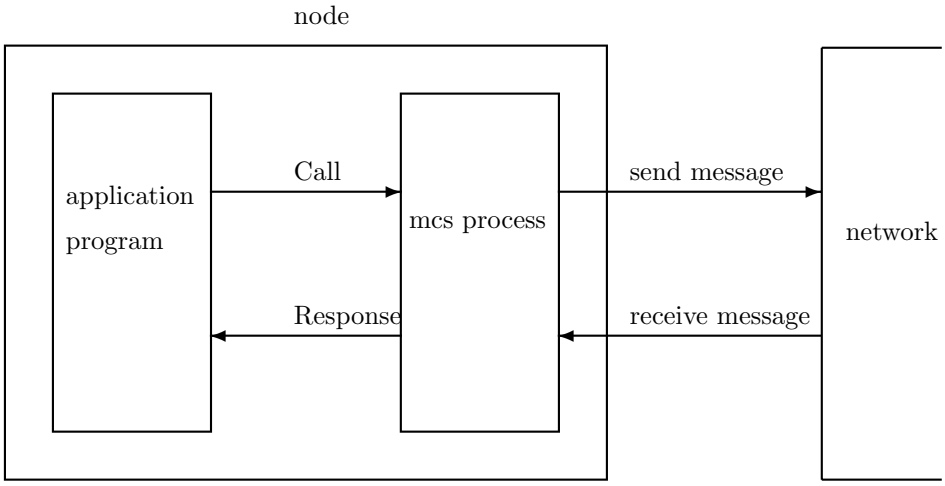


FIG. 1. *System architecture.*

An *mcs process* (or simply *process*) is an automaton with a (possibly infinite) set of states, including an initial state and a transition function. Each interrupt event causes an application of the transition function. The transition function is a function from states and interrupt events to states, sets of response events, and sets of message-send events. That is, the transition function takes as input the current state and an interrupt event, and produces a new state, a set of response events for the application process, and a set of messages to be sent. A *step* of a process p_i is a tuple (s, i, s', R, M), where s and s' are states, i is an interrupt event, R is a set of response events, M is a set of message-send events, and s', R, and M are the result of p's transition function acting on s and i. A *history* of a process p is a mapping h from $\Re$ (real time) to finite (possibly empty) sequences of steps such that

1. for each real time t, there is only a finite number of times $t' < t$ such that the corresponding sequence of steps $h(t')$ is nonempty (thus the concatenation of all the sequences in real time order is a sequence);
2. the old state in the first step is p's initial state; and
3. the old state of each subsequent step is the new state of the previous step.

An *execution* of an mcs is a set of histories, one for each process in P, in which there is a one-to-one correspondence from the messages received by p_i from p_j onto the messages sent by p_j to p_i, for any processes p_i and p_j. An *infinite execution* is an execution in which every history is infinite. We use the message correspondence to define the *delay* of any message in an execution to be the real time of receipt minus the real time of sending. (The network is not explicitly modeled, although the constraints on executions, defined below, imply that the network reliably delivers all messages sent.)

An execution σ is *admissible* if the following conditions hold.

1. For every i and j, every message in σ from p_i to p_j has its delay in the range $[0, d]$, for a fixed nonnegative integer d. (This is a restriction on the network.)
2. For every i, in every prefix of σ the number of calls at p_i can be larger than the number of responses at p_i by at most one. (This is a restriction on the application program.)

Each pair of a *call* event and a subsequent matching *response* event forms an *operation*. The call event marks the start of the operation, while the response event marks the end of that operation. An operation *op* is *invoked* when the application process issues the appropriate call for *op*; *op terminates* when the mcs process issues the appropriate response for *op*. An operation is *pending* if it is invoked and is not terminated. Note that since we assumed that all executions are admissible, there can be at most one pending operation per process; i.e., an application program does not invoke a new operation before the previous one has terminated. Given a particular mcs, an object x implemented by it, and an operation type *op* on x, we denote by $|op(x)|$ the maximum time taken from the invocation to the termination of an operation of type *op* on x in any execution. We denote by $|op|$ the maximum of $|op(x)|$, over all objects x implemented by the mcs.

Every object is assumed to have a *sequential specification* (cf. [41]) defining a set of operations and a set of operation sequences, which are the allowable sequences of operations on that object. For example, in the case of a read/write object, the ordered pair of events $[\text{Read}_i(x), \text{Return}_i(x, v)]$ forms an *operation* for any p_i, x, and v, as does $[\text{Write}_i(x, v), \text{Ack}_i(x)]$. The set of operation sequences consists of all sequences in which every read operation returns the value of the latest preceding write operation (the usual read/write semantics).

A sequence τ of operations for a collection of processes and objects is *legal* if, for every object x, the restriction of τ to operations of x is in the sequential specification of x.

Let τ be an execution or a sequence of operations. Denote by $\tau \mid j$ the restriction of τ to operations invoked by p_j; similarly, denote by $\tau \mid x$ the restriction of τ to operations on object x.

Given an execution σ, let $ops(\sigma)$ be the sequence of call and response events appearing in σ in real time order, breaking ties by ordering all events of the same process in the order in which they appear in that process and then using process ids. An execution σ induces a partial order, $\overset{\sigma}{\longrightarrow}$, on the operations that appear in σ; $op^1 \overset{\sigma}{\longrightarrow} op^2$ if the response event of op^1 appears in $ops(\sigma)$ before the call event of op^2.

Given an execution σ, a sequence of operations τ is a *serialization* of σ if it is a permutation of $ops(\sigma)$. A serialization τ of σ is a *linearization* if it extends $\overset{\sigma}{\longrightarrow}$; that is, if $op^1 \overset{\sigma}{\longrightarrow} op^2$ then $op^1 \overset{\tau}{\longrightarrow} op^2$. In our framework, sequential consistency and linearizability are defined as follows.

DEFINITION 2.1 (sequential consistency). *An execution σ is* sequentially consistent *if there exists a legal serialization τ of σ, such that for each process p_j, $\sigma \mid j = \tau \mid j$.*

DEFINITION 2.2 (linearizability). *An execution σ is* linearizable *if there exists a legal linearization τ of σ, such that for each process p_j, $\sigma \mid j = \tau \mid j$.*

It is possible to mark some operations as *strong*; all other operations are called *weak*. In the case of read/write objects this means that it is possible to use *strong reads* and *strong writes*. We denote the call events for strong operations by $\text{SWrite}(x, v)$ and $\text{SRead}(x)$ and the respective response events by $\text{SAck}(x)$ and $\text{SReturn}(x, v)$.

We now introduce some notation we use in the rest of the paper. By op_i we denote an operation invoked by p_i (weak or strong), and by sop_i we denote a strong operation invoked by process p_i. We use superscripts, e.g., $op_i^1, op_i^2, \ldots$, to distinguish between operations invoked by the same process. We sometimes use a shorthand notation for

read and write operations and denote by $r_i(x, v)$ a weak read operation invoked by process p_i returning v from x; we denote by $w_i(x, v)$ a weak write operation invoked by process p_i writing v to x. Similarly, $sr_i(x, v)$ is a strong read operation invoked by process p_i returning v from x; $sw_i(x, v)$ is a strong write operation invoked by process p_i writing v to x.

3. Definition of hybrid consistency. The following definition requires that (a) strong operations appear to occur in the same order at all processes, and (b) if two operations are invoked by the same process and one of them is strong, then they appear to occur at all processes in the order in which they were invoked.

DEFINITION 3.1 (hybrid consistency based on linearizability). *An execution σ is* hybrid *if there exists a linearization ρ of σ such that for each process p_j, there exists a legal sequence of operations τ_j with the following properties.*

1. *τ_j is a permutation of $ops(\sigma)$.*
2. *If $op_i^1 \xrightarrow{\sigma} op_i^2$ and at least one of op_i^1 and op_i^2 is strong, then $op_i^1 \xrightarrow{\tau_j} op_i^2$ for any i.*
3. *If $op^1 \xrightarrow{\rho} op^2$ and op^1 and op^2 are strong, then $op^1 \xrightarrow{\tau_j} op^2$.*
4. *$\tau_j \mid j = \sigma \mid j$.*

Definition 3.1 defines the view τ_j for each process p_j. This view must include all the memory operations in the execution (this is the first requirement). Any two operations invoked by the same process p_i, where at least one of them is strong, must be viewed by p_j in their order of invocation at p_i (this is the second requirement). The views of all processes must agree on the order of the strong operations, including those by other processes (this is the third requirement). Finally, process p_j must view its own operations in their order of invocation (this is the fourth requirement).

The definition requires the existence of a linearization ρ of σ. However, ρ is used only to force a global order on the strong operations. Therefore, we can require the existence of a linearization only of the strong operations in σ, obtaining an equivalent definition.

When restricted to strong operations, this definition is equivalent to linearizability. However, following the results of [12], we might be interested in examining hybrid models in which strong operations are only sequentially consistent. This leads to the following variant of hybrid consistency.

DEFINITION 3.2 (hybrid consistency based on sequential consistency). *An execution σ is* hybrid *if there exists a serialization ρ of σ such that for each process p_j, there exists a legal sequence of operations τ_j with the following properties.*

1. *τ_j is a permutation of $ops(\sigma)$.*
2. *If $op_i^1 \xrightarrow{\sigma} op_i^2$ and at least one of op_i^1 and op_i^2 is strong, then $op_i^1 \xrightarrow{\tau_j} op_i^2$ for any i.*
3. *If $op^1 \xrightarrow{\rho} op^2$ and op^1 and op^2 are strong, then $op^1 \xrightarrow{\tau_j} op^2$.*
4. *$\tau_j \mid j = \sigma \mid j$.*

The only difference between the two definitions is that the second definition requires that ρ is a serialization of σ rather than a linearization.

3.1. Related definitions. We now present a detailed comparison of hybrid consistency, as defined above, with other definitions of mixed consistency conditions, which distinguish between two kinds of operations.

Most previously known definitions of hybrid consistency are given by specifying, at some degree of formality, *how the mcs handles operations*, i.e., how the hardware

should behave. In contrast, our definition specifies *how the operations appear to the application program*, i.e., how the hardware should appear to the programmer. In particular, our definition is given at the interface between the application program and the mcs, while most previous definitions of hybrid consistency are given at the interface between the mcs and the network.

Although from the architect's point of view it is easier to work with a definition that is given at the interface between the mcs and the network, a definition at the interface between the application program and the mcs is more convenient for programming. It is more difficult to write programs or argue about their correctness when the exact behavior of the hardware has to be considered. Also, programs that are written with respect to an abstract consistency condition are more portable; they can run correctly on different implementations of the consistency condition, regardless of the optimizations that are used by the hardware. Our work complements previous work by presenting an abstract and formal definition at a more comfortable interface, even if this disallows some hardware optimizations.

We now turn to a more detailed comparison of our definition of hybrid consistency with related definitions. It is necessary to make here a distinction between the framework and the definition style we suggest for consistency conditions, and the specific decisions we made in specifying hybrid condition. The current definition of hybrid consistency was given based on our impression of what would be a reasonable model for programming. However, if due to performance reasons, one wishes to change some of these choices, our framework allows such changes to be done very easily. In our opinion, this is one of the main advantages of our framework.

Dubois, Scheurich, and Briggs. The most related condition is *weak ordering*, defined by Dubois, Scheurich, and Briggs [30]. Variables are classified as either regular or synchronization, and it is required that:

1. accesses to global synchronization variables are strongly ordered;
2. no access to a synchronizing variable is issued by a process before all its previous global data accesses have been executed by all the processes;
3. no access to global data is issued by a process before its previous access to a synchronizing variable has been executed by all the processes.

This definition is given in terms of restrictions on the implementation (i.e., at the interface between the mcs processes), rather than on the way the mcs should appear to the programmer (i.e., at the interface between the application programs and the mcs).

This definition rules out implementations which obey the intended semantics of the definition and give better performance. For example, the implementation given in [30] could be improved by pipelining weak operations with strong operations. This improvement seems to obey the intended semantics of the definition, but violates the definition itself. To alleviate this problem and be able to compare this definition with ours, the following definition is specified at the interface between the mcs and the application program, and (we hope) provides the semantics intended by the definition in [30].

DEFINITION 3.3. *Let X be the set of synchronizing variables in the system. An execution σ is* object-based hybrid *if there exists a serialization ρ of σ such that for each process p_j, there exists a legal sequence τ_j of operations such that the following hold.*

1. *τ_j is a permutation of $ops(\sigma)$.*
2. *If $op_i^1 \in (\sigma \mid X)$ or $op_i^2 \in (\sigma \mid X)$ and $op_i^1 \xrightarrow{\sigma} op_i^2$, then $op_i^1 \xrightarrow{\tau_j} op_i^2$.*

3. *If* $op^1, op^2 \in (\sigma \mid X)$ *and* $op^1 \xrightarrow{\rho} op^2$, *then* $op^1 \xrightarrow{\tau_j} op^2$.
4. $\tau_j \mid j = \sigma \mid j$.

This definition is a special case of our definition of hybrid consistency in which strong operations are restricted to special synchronization objects. In contrast, our definition allows the use of strong and weak operations on the same object. This property is exploited in the mutual exclusion algorithm given below in section 4. In this algorithm, the reads from synchronization objects are weak. This results in a reduced number of messages in the system.

Adve and Hill. A different approach splits the responsibility for memory consistency between the software and the hardware. In this approach, the hardware must exhibit some known predefined behavior only for software that obeys certain requirements. For example, the following definition of *weak ordering* was given by Adve and Hill [1]:

> "Hardware is weakly ordered with respect to a synchronization model if and only if it appears sequentially consistent to all software that obeys the synchronization model."

Following this definition, they give a formal definition of a synchronization model DRF0. In this synchronization model, an application program is required to use synchronization operations in any place where a data race[3] is possible. Using the same approach, a generalization of DRF0 that supports further classification of synchronization operations into *releases* and *acquires* is presented in [2, 3]. This synchronization model is called DRF1. Again, the application program is required to use synchronization operations to prevent data races.

The approach taken by Adve and Hill divides the responsibility for correctness between the mcs and the application program: the mcs supports a consistency condition very similar to hybrid consistency, and the application program is required to obey a certain synchronization model, in this case DRF0 (or DRF1). Then it is guaranteed that the programs will run *as if* the mcs was sequentially consistent. In this approach, the mcs provides a guarantee about the results of memory access operations only for programs that obey DRF0 (or DRF1). There is no crisp guarantee (that does not rely on understanding a specific implementation) for programs that violate DRF0 (or DRF1).

Release consistency. *Release consistency*, defined by Gharachorloo et al. [35], is supported in the Stanford DASH multiprocessor. The definition given in [35] assumes three types of operations: *ordinary*, *sync*, and *nsync*. Each type is further divided into reads and writes. The following list of sufficient conditions is then given.

1. Uniprocessor data dependencies are respected.
2. All writes to the same location are serialized in some order and are executed in that order with respect to any process.
3. Before a sync write access is allowed to execute with respect to any other process, all previous ordinary write accesses by the same process must be executed.
4. Sync and nsync accesses are kept sequentially consistent with respect to one another.

Besides being at the interface between the mcs and the network, this definition is given as a list of sufficient conditions on the hardware. These conditions are based on

[3] Informally, a *data race* occurs when the order of two conflicting memory accesses (which are not synchronization operations) is not predetermined; for precise definitions and more discussion, see [1, 9, 11].

the notion of "an operation being executed with respect to a process," whose meaning depends on the specific implementation which was kept in mind when the definition was given.

Formal definitions of release consistency are given in [37, 38].[4] These definitions are quite complex and make reasoning about programs directly with release consistency a very difficult task. To overcome this problem, a synchronization model called PL-programs is defined; this model is similar to DRF1. (In fact, DRF1 is an extension of DRF0, which reflects the release-acquire distinction made by PL.) It is then guaranteed that any program obeying this synchronization model will run as if the mcs was sequentially consistent. This approach, like the one taken by Adve and Hill, gives little support to the design of programs that do not obey specific synchronization models, e.g., our modification of the classical solution to mutual exclusion (presented in the next section). Note that sometimes it is easier to prove the correctness of a program with respect to a consistency condition from scratch than to prove that the program obeys some synchronization model like DRF0, DRF1, or PL.

The major conceptual difference between release consistency and hybrid consistency is the further classification of strong operations into release and acquire. It was shown in [34, 60] that in certain cases, this further classification improves the achievable performance, but in other cases, it does not provide any substantial benefit. It is simple to modify our definition to capture this further distinction (see the definition of asymmetric hybrid consistency in [9]). In addition, release consistency does not require that each process view the weak reads of other processes.

Hybrid consistency, on the other hand, requires that the view of every process include all operations of any other process, in particular, the weak reads. However, our framework allows us to change the definition very easily to omit this requirement; furthermore, none of our results depend on this condition.[5] We believe that this additional requirement makes the programmer's model more comprehensive and therefore more convenient to use, although it may prohibit some otherwise possible hardware optimizations.

4. Programming with hybrid consistency. Hybrid consistency supports a very simple method of programming: use only strong operations and "think sequential consistency" (or linearizability, as the case may be). Clearly, this method is prone to the same performance penalties as sequential consistency. A better programming method is to use weak operations most of the time and strong operations only in those places where global ordering is needed. In this section, we give an example of a mutual exclusion algorithm based on hybrid consistency, and prove its correctness. This example demonstrates that programming with hybrid consistency can be simple and efficient. It also gives some intuition on when strong operations should be used, and how to prove correctness relying on hybrid consistency.

Formally, an algorithm for mutual exclusion consists of four disjoint sections–entry, critical, exit, and remainder (cf. [52]). In the *entry* section, a process tries to gain access to the critical section; the *exit* section is executed by each process upon leaving the critical section; the *remainder* section is the rest of the code. A mutual exclusion algorithm should guarantee

> *mutual exclusion:* no two processes are inside the critical section at the same time;

[4] The definition in [37] is a generalization of the definition in [38] that allows pipelining of memory operations.

[5] The algorithm could be slightly simplified without this condition.

1: $sw(need[i], \text{true})$;
2: $sw(turn, 1-i)$;
3: **if** $r(need[1-i]) = \text{true}$ **and** $r(turn) = 1-i$ **then goto** 3;
$\langle$*critical section*$\rangle$
4: $sw(need[i], \text{false})$;
$\langle$*remainder section*$\rangle$

FIG. 2. *Mutual exclusion using hybrid consistency; code for process* p_i, $i = 0, 1$.

deadlock freedom: in every infinite execution, if there is a process in the entry section, then eventually there is some process in the critical section; and
starvation freedom: in every infinite execution, every process which executes the entry section is eventually granted permission to enter the critical section.

Figure 2 presents a solution for two processes which is fair; it is a simple modification of Peterson's algorithm [50]. The algorithm can be extended to n processes along the lines of Peterson's algorithm. Lines 1–3 are the entry section; line 4 is the exit section. In the code we use the notation $r(x) = v$ to denote a weak read of x returning the value v, and $sw(x, v)$ to denote a strong write of v to x, for any read/write object x and value v.

The lemmas and proofs of this section apply to hybrid consistency based on linearizability and to hybrid consistency based on sequential consistency.

LEMMA 4.1. *The algorithm in Figure* 2 *guarantees mutual exclusion under hybrid consistency.*

Proof. Assume, by way of contradiction, that there is a hybrid execution σ in which p_0 and p_1 execute the critical section together. Denote by $sw_0(need[0], \text{true})$ and $sw_0(turn, 1)$ the last strong writes to $need[0]$ and $turn$ executed by p_0 before entering the critical section, and by $sw_1(need[1], \text{true})$ and $sw_1(turn, 0)$ the last strong writes to $need[1]$ and $turn$ executed by p_1 before entering the critical section. Remember that σ is a hybrid execution and consider a serialization ρ of σ as guaranteed by the definition of hybrid consistency. Without loss of generality, assume that $sw_0(turn, 1)$ appears before $sw_1(turn, 0)$ in ρ. In particular, $sw_0(need[0], \text{true})$ is ordered before $sw_1(turn, 0)$ in ρ. Since p_1 entered its critical section, it follows that p_1's last $r_1(turn)$ before entering returned 1. However, by assumption, $sw_0(turn, 1) \xrightarrow{\rho} sw_1(turn, 0)$. By the definition of τ_1 it follows that $sw_0(turn, 1) \xrightarrow{\tau_1} sw_1(turn, 0) \xrightarrow{\tau_1} r_1(turn, 1)$. Thus, τ_1 is not legal, since $r_1(turn)$ should have returned 0. □

LEMMA 4.2. *The algorithm in Figure* 2 *is free of deadlock under hybrid consistency.*

Proof. Assume, by way of contradiction, that there is a hybrid execution σ in which there is a deadlock. A deadlock occurs if p_0 and p_1 keep executing line 3; that is, p_0 continually reads $r_0(need[1], \text{true})$ and $r_0(turn, 1)$ in line 3 and p_1 continually reads $r_1(need[0], \text{true})$ and $r_1(turn, 0)$.

Since σ is a hybrid execution, there exists a serialization ρ of σ. Without loss of generality, assume $sw_0(turn, 1)$, the write by p_0 immediately preceding p_0's infinite loop, is ordered in ρ before $sw_1(turn, 0)$, the write by p_1 immediately preceding p_1's infinite loop.

Now consider p_0's view, τ_0, as guaranteed by the definition of hybrid consistency. In τ_0, $sw_0(turn, 1)$ precedes $sw_1(turn, 0)$, since τ_0 respects ρ. Let $r_0(turn, 1)$ be the first such read by p_0 that follows $sw_1(turn, 0)$ in τ_0. It exists because there are infinitely many such reads. We now show that no write to *turn* can fall between $sw_1(turn, 0)$ and $r_0(turn, 1)$ in τ_0, violating the legality of τ_0.

Every write to *turn* by p_0 is a strong write and precedes $sw_0(turn, 1)$ in the actual execution, and thus precedes $sw_1(turn, 0)$ in ρ and in τ_0. Every (other) write to *turn* by p_1 is a strong write and precedes $sw_1(turn, 0)$ in the actual implementation, and thus precedes $sw_1(turn, 0)$ in ρ and in τ_0. Thus, $sw_1(turn, 0)$ is the last write to *turn* ordered in τ_0 before $r_0(turn, 1)$, a contradiction to the legality of τ_0. □

LEMMA 4.3. *The algorithm in Figure 2 is starvation free under hybrid consistency.*

Proof. Assume, by way of contradiction, that there exists an infinite hybrid execution σ of the algorithm in which one process, say p_0, is trying to enter the critical section, but is never granted permission. Let $sw_0(turn, 1)$ be the last write of p_0 to *turn*. By Lemma 4.2, p_1 must enter the critical section an infinite number of times. Let ρ be a serialization of all operations in σ, as guaranteed in the definition of hybrid consistency. Every operation in σ must appear in ρ after a finite number of operations. Thus, there exists an operation $sw_1(turn, 0)$ such that $sw_0(turn, 1) \xrightarrow{\rho} sw_1(turn, 0)$. Consider a legal serialization τ_1 of σ, as guaranteed by hybrid consistency. Under the assumption that p_1 enters the critical section infinitely often, τ_1 must include the following subsequence:

$$sw_0(turn, 1) \xrightarrow{\tau_1} sw_1(turn, 0) \xrightarrow{\tau_1} r_1(turn, 1) .$$

This contradicts the assumption that τ_1 is legal. □

The algorithm uses only weak reads in the entry section; hence, when running on an efficient implementation of hybrid consistency (like the one presented in this paper), the entry section is executed faster and with less message overhead.[6]

Note that if the first operation inside the critical section is a weak operation, then we might not have *logical mutual exclusion*. That is, the operations of the entry and exit sections are ordered correctly, but weak operations issued by p_0 inside the critical section may be ordered in the view of p_1 before the last (weak read) operation of the corresponding entry section of p_0. Similarly, weak operations issued by p_1 inside the critical section may be ordered in the view of p_0 before the last (weak read) operation of the corresponding entry section of p_1. This need not happen in every implementation of hybrid consistency, but could happen in some of them. One way to solve this problem is to add one strong (read or write) operation to some object that is not accessed elsewhere in the program, before entering the critical section. This will guarantee that all the operations that are invoked inside the critical section will be ordered after all operations that belong to the entry section. Since the number of operations in the entry section is expected to be quite large, adding one operation (that will be executed only once) should not impose a significant performance loss. A detailed discussion of this problem and another solution can be found in [9].

5. An algorithm for providing hybrid consistency. In this section we present an efficient algorithm for providing hybrid consistency, in both flavors, that supports

[6] Note that by efficient implementations we refer to implementations that execute weak operations faster than the network delay. By the lower bounds proved in [12] and in section 6, strong operations cannot be implemented faster than the network delay.

read/write objects, accessible by either strong or weak operations. The algorithm allows weak operations to be executed instantaneously, while keeping the response time for strong operations proportional to the network delay. Being able to execute weak operations fast is vital for the usefulness of this algorithm, since it is the major advantage hybrid consistency has over sequential consistency. (Recall that in order to implement sequential consistency, the execution time of either reads or writes must be at least linear with the network delay.) Our algorithm assumes that each process holds a complete copy of the entire memory, which allows for instantaneous execution of weak operations, provided that write operations will eventually update all copies. (We later discuss how to relax this assumption.)

The algorithm we describe is completely asynchronous. This means that every process may run at a different rate and that the delay of the network need not be known.

The algorithm uses an atomic broadcast mechanism to send messages. Atomic broadcast is becoming quite common as an underlying tool in the development of distributed systems [16, 17, 47, 53, 55, 57]. Our decision to use atomic broadcast follows this trend and is justified by the immediate benefits that this mechanism provides. An atomic broadcast mechanism guarantees that all messages are delivered at all processes in the same order. Thus, it creates a "global logical time," which simplifies the code of the algorithm and its proof of correctness. Furthermore, using an atomic broadcast mechanism is more modular than implementing the communication protocol directly. Many atomic broadcast algorithms with various degrees of fault tolerance and efficiency have been developed, e.g., [6, 7, 8, 10, 12, 18, 19, 23, 25, 27, 26, 31, 43, 59]. Thus, by employing different atomic broadcast algorithms, one may balance the degree of fault tolerance, message complexity, and time complexity, to fit the desired goals of the implementation.

In the algorithm, weak operations are executed on the local copy of the memory. Strong operations and weak write operations broadcast a message to all processes; an operation is executed by each process when the corresponding message is delivered at the process. Thus, the order of execution should reflect the order in which messages are delivered by the atomic broadcast mechanism.

Unfortunately, there is a problem, since weak operations must return immediately and a weak read that immediately follows a weak write by the same process must return the value written by that write. To handle this problem, we modify the basic idea such that weak writes that are not invoked by the local process are executed under the following conditions. A weak write that is not concurrent[7] with any other (weak or strong) write is always executed when its corresponding message is delivered. If there are two concurrent weak writes, then the one which is delivered first by the atomic broadcast service is executed by every process, while the other one is ignored by every process.

The detailed description of the algorithm and its pseudocode in section 5.1 is followed by a correctness proof in section 5.2 and a complexity analysis in section 5.3.

5.1. The algorithm. We assume a system of n processes, each capable of holding a local copy of the entire memory. The processes are connected by some interconnection network. We assume an atomic broadcast mechanism that supports two

[7] Concurrency is defined with respect to the global order of message delivery. In other words, two weak writes are concurrent if they were invoked by different processes and the same number of messages (that were broadcast to everybody) have been delivered at the invoking processes before they were invoked.

primitives: **abcast** and **asend**. The first primitive, **abcast**, broadcasts a message to all processes atomically. A process may use the second primitive, **asend**, to send a message to a single process, when the interleaving of this message with other messages sent by it, using either **abcast** or **asend**, should reflect their relative order of invocation.

More precisely, let MA be the set of messages sent using **abcast** and denote by MA_i the sequence of messages in MA that are delivered at process p_i, according to the order in which they are delivered at p_i. Then the following hold: (a) for any i and j, $MA_i = MA_j$, and (b) for any two messages $m_i^1, m_i^2 \in MA$ such that both m_i^1 and m_i^2 are sent by p_i and m_i^1 is sent before m_i^2, then m_i^1 is ordered in MA_i before m_i^2. Moreover, let MP_{ij} be the sequence of messages sent from process p_i to process p_j using either **abcast** or **asend**. Then the messages in MP_{ij} are delivered at p_j in the order in which they were sent by p_i.

We use **asend** instead of a regular send to send acknowledgments since the correctness of the algorithm depends on the fact that at each process, the delivery order of *all* messages reflects the order in which they were sent. We use **asend** instead of **abcast** because in some implementations of atomic broadcast, e.g., sequencer/token-based protocols [6, 8, 10, 18, 27, 43, 58], it results in a lower message complexity.

Every process maintains a local copy of the entire memory *mem*, an array of integers *last_mess*, an array of ids *last_id*, an integer counter *received*, and another object *val*, as described below. (See also Figure 3.)

The counter *received* is used to count the messages delivered by the atomic broadcast. It serves as a logical timestamp and its value is added to messages. It is initialized to 0 and incremented each time a message that was sent with **abcast** is delivered.

For each object, *last_mess* and *last_id* uniquely identify the last operation that updated the object. That is, whenever a (weak or strong) write updates an object, *last_mess* is assigned the current value of the variable *received* and *last_id* is assigned the id of the process that invoked the write.

Finally, the variable *val* is used to temporarily store a value that a pending strong read should return, as described later.

Weak operations are executed on the local copy of the memory and return immediately. A weak write causes an `update` message to be broadcast to all processes. This message contains the name of the object to be updated, the new value for the object, and a logical timestamp which is the value of *received* when the operation is invoked. Whenever an `update` message is received, an `ack` message is sent back to the initiator of the operation, using **asend**.

Following this, processes that did not invoke the write execute it on the local copy of the object if either (a) the previous operation that updated the same object was invoked by the same process (held in the variable *last_id*), or (b) the value of *last_mess* is smaller than the value of *received* that is included in the `update` message of the weak write. These conditions guarantee that all writes to the same location are executed by all processes in the same order, based on the following observation: the value *last_mess*$[x]$ is a counter for how many messages were delivered by the atomic broadcast mechanism when object x was last updated. (In that sense the atomic broadcast mechanism serves to order all writes.) If a write w_k has a timestamp bigger than *last_mess*$[x]$ at p_i, then w_k was invoked by p_k after p_k had also seen the last update to x that p_i had seen; call this update w_l. So, w_k causally follows the actual act of updating x which was caused by w_l. In particular, if w_l is a weak write, w_k should not overwrite it. On the other hand, if the timestamp of w_k is smaller than

last_mess[x], then it does not causally follow the actual act of updating x at p_k, and might have overwritten w_l in p_l.

Hence, the first write that passes the timestamp test among all those which are concurrent gets to update the memory. The rest of the concurrent writes fail this test, because we are using the value of *received* when the update message is delivered to update *last_mess*[x], and not the timestamp of the update message itself. So, these writes never get to update any process (except their local process, but even there they are overwritten by the "winner" write), and in a logical sense they are ordered before the "winning" write, which is the first of the concurrent writes to be delivered by the atomic broadcast mechanism.

Note that this order corresponds to the order implied by the logical timestamp of the writes. In other words, it corresponds to the value of the variable *received* at the time of their invocation. Ties among all writes with the same logical timestamp are broken by ordering the first delivered weak write after all other weak writes with the same logical timestamp. The variable *last_id* is required to preserve the order of writes by the same process, since two consecutive weak writes by the same process may have the same logical timestamp.

Whenever a strong operation is invoked, if the operation is a strong read and the last local operation was a weak read, then a `dummy` message is broadcast to all processes. Every process that receives a `dummy` message returns an `ack` message to the initiator using **asend**. Next, the invoking process waits until all `ack` messages for the previous operations and for the current `dummy` message return. Then, a `strong-write` or a `strong-read` message is broadcast to every process. The operation is executed by every process whenever the appropriate message is delivered. A strong write is executed by updating the local copy of the object. A strong read is executed at the invoking process by copying the value of the local copy of the object to *val*. This value is returned when the operation returns. A strong read is executed by other processes by doing nothing. After executing a strong operation, an `ack` message is sent back to the initiator of the operation. A strong operation does not return until all `ack` messages have been received.

A `dummy` message is broadcast between the invocation of a weak read and the invocation of a strong read to guarantee that the invocation of a strong read is separated by at least a certain amount of "logical time" from the invocation of the last previous weak operation by the same process. This allows us to reorder strong reads relative to the delivery of their `strong-read` messages; the exact reordering is described in the correctness proof. Note that there is no need to send a `dummy` message after a weak write, since weak writes broadcast an `update` message; this already guarantees that there is enough "logical time" between the invocation of the weak write and the strong operation.

The precise code appears in Figures 3 and 4; recall that *Ack*, *SAck*, *Return,* and *SReturn* are defined at the end of section 2.

5.2. Proof of correctness. Given an execution σ, we explicitly show how to construct the set of sequences $T = \{\tau_j\}_{j=1}^{n}$, as required in the definition of hybrid consistency. We prove that our algorithm satisfies Definition 3.1; this clearly implies that it also satisfies Definition 3.2.

Very informally, the construction goes as follows. For each process p_j, we first build the sequence τ'_j, consisting of all (weak and strong) write operations, all local weak read operations, and all delivery events. The operations are ordered according to the order in which they occurred in p_j. The delivery events serve as markers

The state of each process p_i consists of the following components:

mem : copy of every object, initially equal to its initial value
last_mess : array of integers with one entry for each object, all initially 0
last_id : array of ids with one entry for each object, all initially 0
received : integer, initially 0
(serial number of the last message delivered to p_i)
missing_acks : integer, initially 0
(number of acknowledgments that are outstanding)
val : an object

Read(x):
generate Return(x, *mem*[x])

Write(x,v):
mem[x] := v
abcast ⟨`update`, x, v, *received*⟩
missing_acks := *missing_acks* + n
generate Ack(x)

SRead(x):
if the last previous operation by p_i was a weak read then
abcast ⟨`dummy`⟩
missing_acks := *missing_acks* + n
endif
wait until *missing_acks* = 0 /* This line is not atomic */
abcast ⟨`strong-read`, x⟩
missing_acks := *missing_acks* + n

SWrite(x,v):
wait until *missing_acks* = 0 /* This line is not atomic */
abcast ⟨`strong-write`, x, v⟩
missing_acks := *missing_acks* + n

FIG. 3. *The algorithm*; *code for process* p_i.

which are useful for the rest of the proof. They are ordered according to the order in which they occurred, such that each delivery event is ordered after the corresponding operation. We show that $T' = \{\tau'_j\}_{j=1}^{n}$ obeys all the requirements in the definition of hybrid consistency, except for including all operations in $ops(\sigma)$ and having delivery events. (This is done in section 5.2.1.)

Following this, we insert the strong reads into T', creating a new set of sequences $T'' = \{\tau''_j\}_{j=1}^{n}$, such that the strong reads are legal. T'' obeys all the requirements in the definition of hybrid consistency, but it does not include all operations of $ops(\sigma)$ and includes the delivery events. This is the most complicated part of the proof, due

```
received ⟨update, x, v, s⟩ from p_j:
    asend ⟨ack, j⟩
    received := received + 1
    if ((s ≥ last_mess[x]) or (last_id[x] = j)) then
        last_mess[x] := received
        last_id[x] := j
        if i ≠ j then
            mem[x] := v
        endif
    endif

received ⟨dummy⟩ from p_j:
    asend ⟨ack, j⟩ >
    received := received + 1

received ⟨strong-write, x, v⟩ from p_j:
    asend ⟨ack, j⟩
    received := received + 1
    last_mess[x] := received
    mem[x] := v
    last_id[x] := j

received ⟨strong-read, x⟩ from p_j:
    asend ⟨ack, j⟩
    received := received + 1
    if (j = i) then val := mem[x] endif

received ⟨ack, i⟩ from p_j:
    missing_acks := missing_acks −1
    if (missing_acks = 0) and there is a pending strong operation to some object x then
        if the strong operation is a write then generate SAck(x)
        else generate SReturn(x,val)
        endif
    endif
```

FIG. 4. *The algorithm*; *code for handling message-receive events at process* p_i.

to the fact that strong reads need to appear consistently in all views. (This is done in section 5.2.2.)

At this stage, we insert in each sequence τ_j'' all weak reads by other processes. This yields a new set of sequences $T''' = \{\tau_j'''\}_{j=1}^{n}$ such that all reads in each τ_j''' are legal. T''' obeys all the requirements in the definition of hybrid consistency except for having the delivery events. (This is done in section 5.2.3.)

Finally, we construct the desired set of sequences T by removing all delivery events from T''', in section 5.2.4.

We now turn to the formal proof and start with some definitions and notations.

For every operation *op* that causes a message to be broadcast, i.e., weak write, strong write, or strong read, we denote by $del_j(op)$ the delivery event of the corre-

sponding message (`update`, `strong-write`, or `strong-read`, accordingly) in p_j. Given a strong read sr_i, we denote by $prev(sr_i)$ the last (`strong-read`, `strong-write`, `update`, or `dummy`) message broadcast by p_i before the `strong-read` message of sr_i. We slightly abuse the notation and denote by $del_j(prev(sr_i))$ the delivery event of $prev(sr_i)$ in p_j.

A read operation $r_i(x, v)$ *reads from a write operation* $w_k(x, v)$ if $w_k(x, v)$ is the last write to x that updates p_i's copy of x before $r_i(x, v)$ reads x. This definition is with respect to the actual execution at p_i, i.e., $\sigma|i$. If $r_i(x, v)$ is strong, then $r_i(x, v)$ is defined to read from x when it copies the value of x into *val*. A read $r_i(x, v)$ is *legal* in a sequence of operations τ if there exists a write $w_k(x, v)$ such that $w_k(x, v) \overset{\tau}{\longrightarrow} r_i(x, v)$ and there does not exist another write $w_l(x, u)$, $u \neq v$, such that $w_k(x, v) \overset{\tau}{\longrightarrow} w_l(x, u) \overset{\tau}{\longrightarrow} r_i(x, v)$; otherwise, the read is *illegal* in τ.

Given a read operation $r_i(x, v)$ that reads from a write $w_k(x, v)$ and a sequence of operations τ, a write $w_l(x, u)$ is an *obliterating write* for $r_i(x, v)$ in τ if $w_k(x, v) \overset{\tau}{\longrightarrow} w_l(x, u)$. Note that in order to define w_k, we need to start with r_i that read from it (in $\sigma|i$).

A *conditional execution interval* of a weak write w_k is the interval of events in σ between the invocation of w_k and the delivery of the `update` message of w_k by the atomic broadcast at p_k. A weak write $w_k(x, v)$ is *overwritten* by another (weak or strong) write $w_j(x, u)$ if $w_j(x, u)$ is performed in p_k during the conditional execution interval of $w_k(x, v)$. A *potential execution interval* of a weak write $w_k(x, v)$ is the largest interval inside the conditional execution interval of w_k, starting with the invocation of $w_k(x, v)$, in which no other write $w_j(x, u)$, $j \neq k$, is performed by p_k. Note that a weak write updates the local copy of the object and returns immediately. The potential execution interval is the interval after a weak write updates the local copy of the object, but may still be overwritten. For example, let $w_k(x, v)$ be a weak write that is overwritten by another write $w_i(x, u)$. If we denote the invocation event of $w_k(x, v)$ by $inv(w_k(x, v))$, we get

$$\overbrace{\underbrace{inv(w_k(x,v)) \ldots del_k(w_i(x,u))}_{\text{potential execution interval}} \ldots del_k(w_k(x,v))}^{\text{conditional execution interval}} .$$

Define the notion of *influence* as follows. A weak write $w_k(x, v)$ *influences* process p_j, $j \neq k$, if it updates the copy of x in p_j (in line 7 of the code for handling `update` messages). A weak write $w_k(x, v)$ *influences* process p_k if it is not overwritten. In particular, not every weak write influences its invoking process. A strong write *influences* every process. Intuitively, a weak write influences its invoking process if it is not overwritten, and influences another process if it is performed by that process. Strong writes are never overwritten and are performed by every process.

Next, define the notion of an operation being *executed* by a process as follows.

- A strong operation sop_k is executed by p_j when the corresponding `strong-write` or `strong-read` message is delivered at p_j.
- A weak operation op_j is executed by p_j when the appropriate call event occurs.
- A weak write w_k that influences p_j, $k \neq j$, is executed by p_j when the corresponding `update` message is delivered at p_j.

Let *abcast order* be the order in which all messages sent with **abcast** are delivered; note that this order is well defined.

We now turn to the details of the proof; for the rest of this section, fix some execution σ of the algorithm.

5.2.1. Creating the initial sequences. For each process p_j, create the sequence of operations and delivery events τ'_j as follows: order all delivery events in p_j according to the order in which they occur in p_j. Next, add all weak operations by p_j according to their order of invocation in p_j (with respect to themselves and to the delivery events in p_j). Next, add all weak writes that are invoked by other processes and are executed by p_j and all strong writes (including the strong writes of p_j) immediately before the delivery of their corresponding `strong-write` or `update` message. Finally, add every weak write w_k that is not executed by p_j immediately before the last write to the same location that is executed in p_j before receiving w'_k's `update` message, breaking ties arbitrarily. Note that if w_k is not executed, then the code of the algorithm guarantees the existence of this write. Note also that all delivery events are ordered in the same order in all sequences τ'_j. That is, for every two operations op^1_k and op^2_l,

$$del_j(op^1_k) \xrightarrow{\tau'_j} del_j(op^2_l) \quad \text{if and only if } del_i(op^1_k) \xrightarrow{\tau'_i} del_i(op^2_l)$$

for every i and j.

The following lemmas show that $\{\tau'_j\}_{j=1}^n$ obey all requirements in the definition of hybrid consistency, except for not including all operations, as follows.

- Legality is shown in Lemma 5.3.
- Condition 1 is not relevant at this point.
- Condition 2 is shown by Lemma 5.5.
- Condition 3 is shown by Lemma 5.1.
- Condition 4 is shown by Lemma 5.2.

Lemma 5.4 concerns a write influencing processes and is used in the proof of Lemma 5.5.

LEMMA 5.1. *There exists a linearization ρ of all the operations in σ such that for every pair of strong operations sop_k and sop_l in τ'_j,*

$$sop_k \xrightarrow{\rho} sop_l \quad \textit{if and only if } sop_k \xrightarrow{\tau'_j} sop_l$$

for every τ'_j.

Proof. The strong operations appear in every τ'_j in an order which is consistent with the abcast order. Since a strong operation does not return before all its `ack` messages return, the real time of its delivery is always between the real time of its invocation and the real time of its termination. Thus, the order in which all strong operations appear in every τ'_j is a linearization. □

LEMMA 5.2. *For every sequence τ'_j, $\tau'_j|j = \sigma'|j$, where σ' is σ without the strong reads.*

Proof. Atomic broadcast delivers all messages sent by the same process in the order in which they were sent. Also, strong operations do not return until all `ack` messages are delivered. Since weak operations are ordered according to their invocation and strong operations are ordered according to their delivery, it follows that $\tau'_j|j = \sigma|j \setminus \{\text{strong reads}\}$ for every sequence τ'_j. □

Note that in each τ'_j, each (local weak) read $r_j(x, v)$ is ordered after the last write to x that is executed by p_j before the invocation of $r_j(x, v)$, with no other write to x in between them. Thus, we have the following lemma.

LEMMA 5.3. *For every sequence τ'_j, all the weak reads of p_j are legal in τ'_j.*

The following lemma is crucial in the rest of the proof.

LEMMA 5.4. *If a write influences some process, then it influences every process.*

Proof. Recall that all messages are delivered to every process in the abcast order. Furthermore, the variable *received* is incremented each time a message that was sent using **abcast** is delivered by the atomic broadcast. Since the atomic broadcast mechanism provides total ordering on the delivery of messages sent with **abcast**, for every weak write, the value of *received* is the same when the corresponding `update` message is delivered in every process. Moreover, this value is different for different weak writes.

Assume, by way of contradiction, that there exists some write that influences some process but not all processes. Let $w_k(x, v)$ be the first such write, i.e., the one for which the value of *received* during the delivery event of its `update` message is minimum. Assume that $w_k(x, v)$ does not influence p_j. Thus, $w_k(x, v)$ is not executed by p_j. The code of the algorithm for executing weak writes implies that there exists another write $w_l(x, u)$ for which the following holds: (a) $w_l(x, u)$ influences p_j, (b) the `update` message of $w_l(x, u)$ is delivered in p_j before the `update` message of $w_k(x, v)$, and (c) the `update` message of $w_l(x, u)$ is delivered in p_k after $w_k(x, v)$ is invoked. (This is the condition which is checked whenever an `update` message is delivered.) Since $w_k(x, v)$ is the first write that influences only some of the processes, $w_l(x, u)$ influences p_k and therefore $w_k(x, v)$ is overwritten.

Atomic broadcast delivers all messages to all processes in the same order. Since $w_k(x, v)$ is the first write that influences only some of the processes, $w_l(x, u)$ influences all processes and its `update` or `strong-write` message is delivered before the `update` message of $w_k(x, v)$ to all processes. Thus, by the assumptions about $w_l(x, u)$, none of the processes executes $w_k(x, v)$. Hence, $w_k(x, v)$ does not influence any process. This is a contradiction to the assumption that $w_k(x, v)$ influences some process. □

LEMMA 5.5. *For each sequence of operations τ'_j and every pair of operations op_k^1 and op_k^2 in τ'_j such that either op_k^1 or op_k^2 is strong,*

$$op_k^1 \xrightarrow{\tau'_j} op_k^2 \quad \textit{if and only if } op_k^1 \xrightarrow{\sigma} op_k^2.$$

Proof. By Lemma 5.2, the claim holds if $k = j$. Thus, for the rest of the proof, we may assume that $k \neq j$.

Assume, by way of contradiction, that for some sequence τ'_j and two operations op_k^1 and op_k^2 such that either op_k^1 or op_k^2 is strong,

$$op_k^1 \xrightarrow{\tau'_j} op_k^2 \quad \text{but } op_k^2 \xrightarrow{\sigma} op_k^1.$$

We claim that op_k^1 is a weak write that does not influence p_j and op_k^2 is strong. Strong operations and weak writes that influence p_j are executed according to the abcast order. This order is consistent with the order by which operations are invoked at each process. Therefore, the only operations that may not be ordered correctly in τ'_j with respect to the previous and next strong operations by the same process are weak writes that do not influence p_j. Moreover, since strong operations are ordered immediately before their corresponding delivery events, and since weak writes are always ordered before their corresponding delivery event, a weak write that does not influence p_j may not be ordered after a later strong operation by the same process.

Assume, without loss of generality, that op_k^1 writes to x. Let w_i be the first write to x that influences p_j and is ordered after op_k^1 in τ_j'. The existence of such a write is guaranteed by the assumption that op_k^1 does not influence p_j and the construction of τ_j'; w_i is the write that caused op_k^1 to be ordered before op_k^2. Note that if w_i did not exist, op_k^1 would have been executed and would have been ordered where its delivery event is ordered, and therefore also after op_k^2. Since op_k^1 is inserted before op_k^2 in τ_j due to w_i, and since by construction, w_i is ordered next to its delivery event, it follows that w_i is delivered in p_j before op_k^2.

If $i \neq j$, or if $i = j$ and op_k^2 is delivered after the completion of the conditional execution interval of w_i in p_j, then by Lemma 5.4, w_i influences every process. Thus, w_i is delivered in every process before op_k^2. In particular, w_i is delivered in p_k before op_k^2. Since op_k^2 is a strong operation and since op_k^1 is invoked after op_k^2, w_i is delivered in p_k before op_k^1 starts. Since op_k^1 is inserted before its delivery event, there exists another write $w_l(x, w)$ that influences p_j and is delivered in p_j between w_i and op_k^1. Thus, $w_l(x, w)$ is ordered in τ_j' between w_i and op_k^1. This contradicts the assumption that w_i is the first write to x that influences p_j and is ordered after op_k^1 in τ_j'.

If $i = j$ and op_k^2 is delivered inside the conditional execution interval of w_i, then the `ack` message of op_k^2 is delivered in p_k after the `update` message of w_i. By Lemma 5.4, w_i influences p_k and is delivered in p_k before op_k^1 starts. Therefore, there exists another write $w_l(x, v)$ that influences p_j and is delivered in p_j between w_i and op_k^1. Thus, $w_l(x, w)$ is ordered in τ_j' between w_i and op_k^1. This contradicts the assumption that w_i is the first write to x that influences p_j and is ordered after op_k^1 in τ_j'. □

Thus, we have shown that the sequences $\{\tau_j'\}_{j=1}^n$ obey all the requirements in the definition of hybrid consistency; however, it does not include all the operations of $ops(\sigma)$ and includes delivery events. To finish the proof, we must insert the missing operations, i.e., the missing reads, to each τ_j' without violating the requirements of the definition of hybrid consistency.

5.2.2. Inserting the strong reads. We start with the strong reads. For each j, create τ_j'' by inserting the strong reads into τ_j' according to the abcast order of their `strong-read` messages. Unlike strong writes, sometimes it is not possible to insert a strong read immediately before its corresponding delivery event and still be legal. This is because the `strong-read` message can be delivered at another process inside a potential execution interval of a weak write to the same object. If this happens, then the last value written to the object is the value written by the local weak write, and not the value returned by the strong read. Therefore, in order to maintain the legality of the sequence, the strong read has to be inserted before the weak write. However, since hybrid consistency requires that all views of all processes agree on the order of strong operations, if a strong read $sr_i(x, v)$ is inserted in one sequence before another strong operation sop_k, then $sr_i(x, v)$ must be inserted before sop_k in every sequence τ_j''. On the other hand, if sop_k is a strong read that is inserted long before its corresponding `strong-read` message, then inserting $sr_i(x, v)$ before sop_k could cause $sr_i(x, v)$ to be ordered before a previous weak operation by p_i. In order to prevent such chain reactions, it is sometimes required to drag a previously inserted strong read $sr_k(y, u)$ before $sr_i(x, v)$ instead of inserting $sr_i(x, v)$ before $sr_k(y, u)$. For example, if we have

$$\tau_j'' = \ldots w_j(x, u) \ldots sop_q \ldots sr_k(y, u) \ldots del_j(sr_i(x, v)) \ldots del_j(w_j(x, u)) \text{ and}$$

$$\tau_l'' = \ldots sop_q \ldots sr_k(y, u) \ldots del_l(sr_i(x, v)),$$

then $sr_i(x,v)$ must be inserted before $w_j(x,u)$ and sop_q in τ_j''. Moreover, $sr_i(x,v)$ must be inserted before sop_q in τ_l'', too. In addition, if $sr_k(y,u)$ is a strong read that is dragged by the insertion of $sr_i(x,v)$, then $sr_k(y,u)$ must be ordered before $sr_i(x,v)$ and sop_q in both τ_j'' and τ_l''.

To insert the strong reads, we introduce some definitions. For each strong read $sr_i(x,v)$, and for each process p_j, if the `strong-read` message of $sr_i(x,v)$ is delivered inside a potential execution interval of some weak write $w_j(x,u)$, then let $I_j(w_j(x,u), sr_i(x,v))$ be the set of strong operations already ordered in τ_j' between the invocation of $w_j(x,u)$ and the delivery of the `strong-read` message of $sr_i(x,v)$. Let $B_j(w_j(x,u), sr_i(x,v))$ be the set of strong reads $\{sr_q\}$ such that $sr_q \in I_j(w_j(x,u), sr_i(x,v))$ and the write sr_q reads from is included in the potential execution interval of $w_j(x,u)$. Let $I_j(sr_i(x,v))$ be the union of $I_j(w_j(x,u), sr_i(x,v))$ over all such $w_j(x,u)$; let $B_j(sr_i(x,v))$ be the union of $B_j(w_j(x,u), sr_i(x,v))$, over all such $w_j(x,u)$. Let $B(sr_i(x,v)) = \cup_{j=1}^{n} B_j(sr_i(x,v))$ and $I(sr_i(x,v)) = \cup_{j=1}^{n} I_j(sr_i(x,v))$, and let $C(sr_i(x,v))$ be the set of strong reads in $I(sr_i(x,v)) \setminus B(sr_i(x,v))$ and $D(sr_i(x,v)) = I(sr_i(x,v)) \setminus C(sr_i(x,v))$.

Intuitively, $I(sr_i(x,v))$ is the set of strong operations that $sr_i(x,v)$ might be inserted before, although $del_j(sr_i(x,v))$ is ordered after them in some sequence τ_j'. $D(sr_i(x,v))$ is the set of strong operations that $sr_i(x,v)$ is actually inserted before, although $del_j(sr_i(x,v))$ is ordered after them in some sequence τ_j'. $C(sr_i(x,v))$ is the set of strong reads that might be dragged by $sr_i(x,v)$, while $B(sr_i(x,v))$ is the set of strong reads that appear in $I(sr_i(x,v))$ and are not dragged by $sr_i(x,v)$. $B(sr_i(x,v))$ is merely used to define $C(sr_i(x,v))$. For example, if we have

$$\tau_j'' = \ldots w_j(x,v), sw_q, sr_k(y,u), w_g(z,w), sr_l(z,w), del_j(sr_i(x,u)), del_j(w_j(x,v)), \ldots$$

and this is the only potential execution interval of a weak write to x that includes the delivery event of $sr_i(x,u)$, then $I(sr_i(x,u)) = \{sw_q, sr_k(y,u), sr_l(z,w)\}$, $B(sr_i(x,u)) = \{sr_l(z,w)\}$, $C(sr_i(x,u)) = \{sr_k(y,u)\}$, and $D(sr_i(x,u)) = \{sw_q, sr_l(z,w)\}$.

Add $sr_i(x,v)$ to each τ_j' in the last possible place such that it will be ordered before every strong operation in $D(sr_i(x,v))$, before the delivery event of its `strong-read` message and before any obliterating write operation. For every strong read sr_q in $C(sr_i(x,v))$, if $sr_i(x,v)$ is ordered before sr_q in τ_j', then reorder sr_q immediately before $sr_i(x,v)$ in τ_j'. In this case, we say that sr_q is *dragged* by $sr_i(x,v)$ in τ_j'. If there is more than one strong read that is dragged by $sr_i(x,v)$, then break ties according to the abcast order of their corresponding `strong-read` messages.

The following lemmas show that $\{\tau_j''\}_{j=1}^{n}$ obey all requirements in the definition of hybrid consistency, except for not including all operations, as follows.

- Legality follows from legality of $\{\tau_j'\}_{j=1}^{n}$ and Lemma 5.7.
- Condition 1 is not relevant at this point.
- Condition 2 follows from the fact that $\{\tau_j'\}_{j=1}^{n}$ obeys it and Lemma 5.11.
- Condition 3 follows from the fact that $\{\tau_j'\}_{j=1}^{n}$ obeys it and Lemma 5.9.
- Condition 4 follows from the fact that $\{\tau_j'\}_{j=1}^{n}$ obeys it and Lemma 5.10.

Lemma 5.6 gives more technical information concerning the influence of a write and is used in some of the lemmas' proofs. Lemma 5.8 is another technical lemma.

LEMMA 5.6. *A strong read reads from a write that influences every process.*

Proof. Let sr_i be a strong read that reads from some write w_k. If w_k is strong, then by definition, w_k influences every process. So, assume that w_k is weak. If $k = i$, then since sr_i is a strong operation, it is not executed until the conditional execution interval of w_k is completed. Therefore, w_k is not overwritten, and by Lemma 5.4,

w_k influences every process. If $k \neq i$, then w_k influences p_i, and by Lemma 5.4, w_k influences every process. □

LEMMA 5.7. *Every strong read is legal in every sequence* τ_j''.

Proof. We prove that every strong read is ordered in every sequence τ_j'' after the write it reads from. Since, by definition, every strong read appears in every sequence τ_j'' before any obliterating write operation, this will imply that the strong read is legal.

Assume, by way of contradiction, that there exists a strong read that is ordered in some sequence τ_j'' before the write it reads from. Let $sr_i(x,v)$ be the first strong read such that, following the insertion of $sr_i(x,v)$, there exists a strong read which is ordered in some sequence τ_j'' before the write it reads from. Thus, either $sr_i(x,v)$ is ordered in τ_j'' before the write it reads from or there exists another strong read $sr_s(y,w)$, which is dragged in τ_j'' before the write it reads from.

Assume that $sr_i(x,v)$ is ordered in τ_j'' before the write it reads from, and denote this write by $w_k(x,v)$. In particular, $sr_i(x,v)$ is ordered before $del_j(w_k(x,v))$ in τ_j''. Thus, $del_l(sr_i(x,v))$ is ordered in some sequence τ_l'' inside a potential execution interval of some weak write $w_l(x,u)$ that includes a strong operation sop_q which is ordered in τ_j'' before $del_j(w_k(x,v))$. (If there is no such sop_q, then there is no reason to order $sr_i(x,v)$ in τ_j'' before the write it reads from.) By the assumption that $sr_i(x,v)$ reads from $w_k(x,v)$ and by Lemma 5.6, $w_k(x,v)$ influences every process. Therefore, $del_l(w_k(x,v))$ is ordered in τ_l'' before the potential execution interval of $w_l(x,u)$. Since we assumed that sop_q is ordered in τ_j'' before $del_j(w_k(x,v))$, and since $del_j(sop_q)$ is ordered after $del_j(w_k(x,v))$, it follows that sop_q is a strong read and the write sop_q reads from is ordered inside the potential execution interval of $w_l(x,u)$. Denote the write sop_q reads from by w_m. By Lemma 5.6, both w_m and $w_k(x,v)$ influence every process, and therefore both w_m and $w_k(x,v)$ are ordered immediately before their delivery events. Thus, w_m is ordered after $w_k(x,v)$ in τ_j''. By the minimality of $sr_i(x,v)$, sop_q is ordered after w_m in τ_j'', and therefore after $w_k(x,v)$. This is a contradiction.

Assume that $sr_s(y,w)$ is dragged by $sr_i(x,v)$ before the write it reads from, and denote this write by $w_k(y,w)$. In particular, $sr_s(y,w)$ is dragged by $sr_i(x,v)$ before $del_j(sr_s(y,w))$ in τ_j''. By the minimality of $sr_i(x,v)$, $sr_s(y,w)$ is ordered after $w_k(y,w)$ in every sequence τ_p'' before the insertion of $sr_i(x,v)$. Thus, $del_l(sr_i(x,v))$ and $del_l(sr_s(y,w))$ are ordered inside a potential execution interval of some weak write $w_l(x,u)$ that includes a strong operation sop_q which is ordered in τ_j'' before $del_j(w_k(y,w))$. Since we assumed that sop_q is ordered in τ_j'' before $del_j(w_k(y,w))$, and since $del_j(sop_q)$ is ordered after $del_j(w_k(y,w))$, it follows that sop_q is a strong read and the write sop_q reads from is ordered inside the potential execution interval of $w_l(x,u)$. Denote the write sop_q reads from by w_m. By Lemma 5.6, both w_m and $w_k(x,v)$ influence every process, and therefore both w_m and $w_k(x,v)$ are ordered immediately before their delivery events. Thus, w_m is ordered after $w_k(x,v)$ in τ_j''. By the minimality of $sr_i(x,v)$, sop_q is ordered after w_m in τ_j'', and therefore after $w_k(y,w)$. This is a contradiction. □

LEMMA 5.8. *Every strong read* $sr_i(x,v)$ *is ordered after* $del_j(prev(sr_i(x,v)))$ *in every sequence* τ_j''.

Proof. Assume, by way of contradiction, that there exists a strong read $sr_i(x,v)$ that is ordered in some sequence τ_j'' before $del_j(prev(sr_i(x,v)))$. Assume, without loss of generality, that $sr_i(x,v)$ is the first strong read such that following the insertion of $sr_i(x,v)$, there exists a strong read $sr_s(y,w)$ which is ordered in some sequence τ_j'' before $del_j(prev(sr_s(y,w)))$. Thus, either $sr_i(x,v)$ is ordered in τ_j'' before

$del_j(prev(sr_i(x,v)))$ or there exists another strong read $sr_s(y,w)$ which is dragged in τ_j'' before $del_j(prev(sr_s(y,w)))$.

Assume that $sr_i(x,v)$ is ordered in τ_j'' before $del_j(prev(sr_i(x,v)))$. Thus, there exists a sequence τ_l'' for which $del_l(sr_i(x,v))$ is ordered in τ_l'' inside a potential execution interval of some weak write $w_l(x,u)$ that includes a strong operation sop_q which is ordered in τ_j'' before $del_j(prev(sr_i(x,v)))$. Since $sr_i(x,v)$ does not return until all `ack` messages of $prev(sr_i(x,v))$ are delivered, $del_l(prev(sr_i(x,v)))$ is ordered in τ_l'' before the potential execution interval of $w_l(x,u)$. Since sop_q is ordered before $del_j(prev(sr_i(x,v)))$ in τ_j'' and after $del_l(prev(sr_i(x,v)))$ in τ_l'', then sop_q is a strong read and the write sop_q reads from is ordered inside the potential execution interval of $w_l(x,u)$. Denote the write sop_q reads from by w_m. By Lemma 5.6, w_m influences every process and is therefore ordered in τ_j'' immediately before $del_j(w_m)$. Thus, w_m is ordered after $del_j(prev(sr_i(x,v)))$ in τ_j''. By Lemma 5.7, sop_q is ordered after w_m in τ_j'', and therefore after $del_j(prev(sr_i(x,v)))$. This is a contradiction.

Assume that $sr_s(y,w)$ is dragged by $sr_i(x,v)$ before $del_j(prev(sr_s(y,w)))$ in τ_j''. By the minimality of $sr_i(x,v)$, $sr_s(y,w)$ is ordered after $del_p(prev(sr_s(y,w)))$ in every sequence τ_p'' before the insertion of $sr_i(x,v)$. Thus, there exists a sequence τ_l'' for which $del_l(sr_i(x,v))$ and $del_l(sr_s(y,w))$ are ordered inside a potential execution interval of some weak write $w_l(x,u)$ that includes a strong operation sop_q which is ordered in τ_j'' before $del_j(prev(sr_s(y,w)))$. Since sop_q is ordered before $del_j(prev(sr_s(y,w)))$ in τ_j'' and after $del_l(prev(sr_s(y,w)))$ in τ_l'', it follows that sop_q is a strong read and the write sop_q reads from is ordered inside the potential execution interval of $w_l(x,u)$. Denote the write sop_q reads from by w_m. By Lemma 5.6, w_m influences every process, and therefore is ordered in τ_j'' immediately before $del_j(w_m)$. Thus, w_m is ordered after $del_j(prev(sr_s(y,w)))$ in τ_j''. By Lemma 5.7, sop_q is ordered after w_m in τ_j'', and therefore after $del_j(prev(sr_s(y,w)))$. This is a contradiction. □

LEMMA 5.9. *There exists a linearization ρ of all the operations in σ such that for every pair of strong operations sop_k and sop_l,*

$$sop_k \xrightarrow{\rho} sop_l \quad \textit{if and only if } sop_k \xrightarrow{\tau_j''} sop_l \textit{ for every } \tau_j''.$$

Proof. We have to show that all the strong operations are ordered in all the sequences τ_j'' in the same order, and that this order is a linearization of the strong operations in $ops(\sigma)$.

We first show that all strong operations appear in the same order in all sequences τ_j''. Assume, by way of contradiction, that there exist two strong operations sop_k and sop_i and two sequences τ_j'' and τ_l'' such that

$$sop_k \xrightarrow{\tau_j''} sop_i \quad \text{but } sop_i \xrightarrow{\tau_l''} sop_k.$$

By Lemma 5.1 and since strong writes are not dragged by strong reads, this can only happen if at least one of sop_k or sop_i is a strong read.

If both sop_k and sop_i are strong reads, then let $sop_i = sr_i(x,v)$ and $sop_k = sr_k(y,w)$. Assume, without loss of generality, that $sr_i(x,v)$ appears before $sr_k(y,w)$ in the order of insertion for strong reads. Thus, the `strong-read` message of $sr_k(y,w)$ is delivered after the `strong-read` message of $sr_i(x,v)$. Since

$$sr_k(y,w) \xrightarrow{\tau_j''} sr_i(x,v),$$

then either $sr_i(x,v) \in D(sr_k(y,w))$ or $sr_k(y,w)$ is dragged by another strong read before sr_i. In either case, $sr_k(y,w)$ is ordered before $sr_i(x,v)$ in every sequence τ_l''. This is a contradiction.

Otherwise, without loss of generality, sop_i is a strong read and sop_k is a strong write. Let $sop_i = sr_i(x,v)$. If the `strong-read` message of $sr_i(x,v)$ is delivered before the `strong-write` message of sop_k, then $sr_i(x,v)$ is ordered before sop_k in every sequence τ_j''. Therefore, the `strong-read` of $sr_i(x,v)$ is delivered after the `strong-write` message of sop_k. Since $sr_i(x,v)$ is ordered before sop_k in τ_j'', then either $sop_k \in D(sr_i(x,v))$ or $sr_i(x,v)$ is dragged by another strong read before sop_k. In any case, $sr_i(x,v)$ is ordered before sop_k in every sequence τ_l''. This is a contradiction.

We now show that the order in which strong operations appear in τ_j'' preserves the order implied by ρ. Assume, by way of contradiction, that the order in which all strong operations appear in every sequence τ_j'' is not a linearization. Note that strong reads are always inserted before their corresponding delivery event. By Lemma 5.1 and since strong writes are not dragged by strong reads, the total order is not a linearization only if there exist a strong read $sr_i(x,v)$ and another strong operation sop_k such that sop_k terminates before $sr_i(x,v)$ begins, but

$$sr_i(x,v) \xrightarrow{\tau_j''} sop_k$$

for every sequence τ_j''. Since, by definition,

$$sop_k \xrightarrow{\tau_j''} del_j(sop_k)$$

for every sequence τ_j'', this can only happen in one of two cases.

Case 1. The delivery event of the `strong-read` message of $sr_i(x,v)$ is ordered inside the potential execution interval of some $w_l(x,v)$, and sop_k is also ordered inside the same potential execution interval. Then the `ack` message for sop_k is delivered in p_k after the `update` message of $w_l(x,u)$, while the `strong-read` message of $sr_i(x,v)$ is delivered in all processes before the `update` message of $w_l(x,u)$. Hence, the `strong-read` message of $sr_i(x,v)$ is delivered in p_k before sop_k terminates. This contradicts the assumption that sop_k terminates before $sr_i(x,v)$ begins.

Case 2. $sr_i(x,v)$ is dragged by some other strong read $sr_q(y,w)$ before sop_k. Thus, the `strong-read` message of $sr_q(y,w)$ is delivered inside the potential execution interval of some weak write $w_l(y,z)$ that includes both $del_l(sop_k)$ and $del_l(sr_i(x,v))$. Therefore, the `ack` message for sop_k is delivered in p_k after the `update` message of $w_l(y,z)$, while the `strong-read` message of $sr_i(x,v)$ is delivered in all processes before the `update` message of $w_l(y,z)$. Hence, the `strong-read` message of $sr_i(x,v)$ is delivered in p_k before sop_k terminates. This contradicts the assumption that sop_k terminates before $sr_i(x,v)$ begins. □

LEMMA 5.10. *For every sequence* τ_j'', $\tau_j''|j = \sigma|j$.

Proof. Consider any two operations op_j^1 and op_j^2 such that $op_j^1 \xrightarrow{\sigma} op_j^2$. We show that

$$op_j^1 \xrightarrow{\tau_j} op_j^2$$

by a simple case analysis. If neither op_j^1 nor op_j^2 is a strong read, then the claim holds by Lemma 5.2 and the fact that the insertion of strong reads does not change the order of other operations. If both op_j^1 and op_j^2 are strong reads, then the claim holds

by Lemma 5.9. If only op_j^1 is a strong read, then the claim holds because strong reads are inserted before their corresponding delivery event. If, on the other hand, only op_j^2 is a strong read, then by Lemma 5.8,

$$del_j(prev(op_j^2)) \xrightarrow{\tau_j''} op_j^2.$$

By construction of τ_j'',

$$op_j^1 \xrightarrow{\tau_j''} del_j(prev(op_j^2)).$$

Thus,

$$op_j^1 \xrightarrow{\tau_j''} op_j^2$$

and the claim holds. (Note that we have covered all the cases, since we still do not have weak reads by other processes.) □

LEMMA 5.11. *For each sequence of operations τ_j'' and every pair of operations op_k^1 and op_k^2 in τ_j'' such that either op_k^1 or op_k^2 is strong,*

$$op_k^1 \xrightarrow{\tau_j''} op_k^2 \quad \textit{if and only if } op_k^1 \xrightarrow{\sigma} op_k^2.$$

Proof. By Lemma 5.10, the claim holds for $k = j$. Thus, we assume for the rest of the proof that $k \neq j$. Assume, by way of contradiction, that for some sequence τ_j'' and two operations op_k^1 and op_k^2 such that either op_k^1 or op_k^2 is strong,

$$op_k^1 \xrightarrow{\tau_j''} op_k^2 \quad \text{but } op_k^2 \xrightarrow{\sigma} op_k^1.$$

By Lemmas 5.5 and 5.9 and the construction of τ_j'', this can only happen in one of the following cases.

Case 1. op_k^1 is a strong read and op_k^2 is a weak write. By the construction of τ_j'', op_k^2 is ordered before $del_j(op_k^2)$. In particular, op_k^2 is ordered in τ_j'' before $del_j(prev(op_k^1))$. By Lemma 5.8, op_k^1 is ordered in τ_j'' after $del_j(prev(op_k^1))$. Thus,

$$op_k^2 \xrightarrow{\tau_j''} op_k^1.$$

This is a contradiction.

Case 2. op_k^2 is a strong read and op_k^1 is a weak write that does not influence p_j. By definition, op_k^2 is ordered before $del_p(op_k^2)$ in every sequence τ_p''. Assume, without loss of generality, that op_k^1 writes to x, and let $w_l(x, w)$ be the last write that influences p_j before the delivery of the `update` message of op_k^1. Thus, op_k^1 is invoked before the delivery of the `update` or `strong-write` message of $w_l(x, w)$ in p_k. Moreover, op_k^1 is ordered in τ_j'' before op_k^2 because $w_l(x, v)$ is ordered in τ_j'' before op_k^2. Since op_k^2 does not return before all of its `ack` messages are delivered, and since every `ack` message is sent using **asend**, the `ack` message for op_k^2 is delivered in p_k after the `update` or `strong-write` message of $w_l(x, w)$. Thus, op_k^1 is invoked after the delivery of the `update` or `strong-write` message of $w_l(x, w)$. This is a contradiction. □

5.2.3. Inserting weak reads. We now insert the weak reads. A sequence τ_j''' is constructed from each sequence τ_j'', by inserting into τ_j'' all the weak reads that are invoked by other processes than p_j, one after the other as follows: every weak read $r_k(x,v)$, $k \neq j$, is inserted in the first possible place such that it will be ordered after the previous strong operation by p_k (if there is one) and after the write it reads from (if there is one). Note that this is well defined: if there is no previous strong operation and the weak read returns the initial value of the object, then the weak read is inserted at the beginning of the sequence.

The following lemmas show that $\{\tau_j'''\}_{j=1}^n$ obey all requirements in the definition of hybrid consistency, except for including delivery events, as follows.

- Legality follows from legality of $\{\tau_j''\}_{j=1}^n$ and Lemma 5.13.
- Condition 1 follows from the construction.
- Condition 2 follows from the fact that $\{\tau_j''\}_{j=1}^n$ obeys it and Lemma 5.12.
- Condition 3 follows from the fact that $\{\tau_j''\}_{j=1}^n$ obeys it and that adding weak reads does not affect it.
- Condition 4 follows from the fact that $\{\tau_j''\}_{j=1}^n$ obeys it and that adding weak reads does not affect it.

LEMMA 5.12. *For each sequence of operations τ_j''' and every pair of operations op_k^1 and op_k^2 such that either op_k^1 or op_k^2 is strong,*

$$op_k^1 \xrightarrow{\tau_j'''} op_k^2 \quad \textit{if and only if } op_k^1 \xrightarrow{\sigma} op_k^2.$$

Proof. Assume, by way of contradiction, that there exist a sequence τ_j''' and two operations op_k^1 and op_k^2 such that either op_k^1 or op_k^2 is strong,

$$op_k^1 \xrightarrow{\tau_j'''} op_k^2 \quad \text{but } op_k^2 \xrightarrow{\sigma} op_k^1.$$

By Lemma 5.10, the claim of the lemma holds if $k = j$. By Lemma 5.11 and since the insertion of weak reads does not change the order of other operations in τ_j''', the claim of the lemma holds if neither op_k^1 nor op_k^2 is a weak read. Moreover, since weak reads are inserted after the previous strong operation by the same process, the only case in which op_k^1 may be ordered in τ_j''' before op_k^2 is when op_k^2 is a weak read, op_k^1 is a strong operation, $k \neq j$, and the write op_k^2 reads from is ordered after op_k^1 in τ_j'''.

Assume, without loss of generality, that op_k^2 reads from $w_l(x,v)$. Lemma 5.11 and the assumption that op_k^2 reads from $w_l(x,v)$ imply that $l \neq k$. Thus, $w_l(x,v)$ influences every process and is therefore ordered in τ_j''' immediately before $del_j(w_l(x,v))$. If op_k^1 is a strong write, then it is ordered in τ_j''' immediately before $del_j(op_k^1)$. Thus,

$$del_j(op_k^1) \xrightarrow{\tau_j'''} w_l(x,v) \xrightarrow{\tau_j'''} del_j(w_l(x,v)).$$

If, on the other hand, op_k^1 is a strong read, then by Lemma 5.8, op_k^1 is ordered in τ_j''' after $del_j(prev(op_k^1))$. Thus,

$$del_j(prev(op_k^1)) \xrightarrow{\tau_j'''} w_l(x,v) \xrightarrow{\tau_j'''} del_j(w_l(x,v)).$$

In either case, by the use of atomic broadcast, the `update` or `strong-write` message of $w_l(x,v)$ is delivered in p_k after op_k^2 is executed. This contradicts the assumption that op_k^2 reads from $w_l(x,v)$. ∎

LEMMA 5.13. *Every read is legal in every sequence* τ_j'''.

Proof. By Lemma 5.7, all strong reads are legal in any of the sequences τ_j'''. Thus, assume, by way of contradiction, that there exist a sequence τ_j''' and a weak read $r_i(x,v)$ such that $r_i(x,v)$ is not legal in τ_j'''. Denote by $w_l(x,v)$ the write $r_i(x,v)$ reads from and by sop_i the next strong operation by p_i. By Lemma 5.3 and since the insertion of (strong or weak) read operations does not change the order of write operations (by any process) or the order of weak reads by p_j, $i \neq j$. Thus, by the rules for inserting weak reads, there exists an obliterating write $w_k(x,u)$ such that

$$w_l(x,v) \xrightarrow{\tau_j'''} w_k(x,u) \xrightarrow{\tau_j'''} sop_i.$$

If $k = j$, then the `update` or `strong-write` message of $w_k(x,u)$ is delivered at p_i after the execution of $w_l(x,v)$ and before the delivery of the `ack` message of sop_i. Thus, either $w_k(x,u)$ or another write to x is executed by p_i between the execution of $w_l(x,v)$ and the invocation of $r_i(x,v)$. This contradicts the assumption that $r_i(x,v)$ reads from $w_l(x,v)$.

Thus, $k \neq j$. We may assume, without loss of generality, that $w_k(x,u)$ influences p_j. Otherwise, there exists another write $w_q(x,w)$ that influences p_j such that

$$w_l(x,v) \xrightarrow{\tau_j'''} w_k(x,u) \xrightarrow{\tau_j'''} w_q(x,w) \xrightarrow{\tau_j'''} sop_i,$$

and we could have chosen $w_q(x,w)$. Thus, $w_k(x,u)$ is ordered in τ_j''' immediately before $del_j(w_k(x,u))$. By Lemma 5.4, $w_k(x,u)$ influences every process. By the construction of τ_j''', sop_i is ordered before $del_j(sop_i)$. Thus, the `update` or `strong-write` message of $w_k(x,u)$ is delivered at p_i after the execution of $w_l(x,v)$ and before the delivery of the `strong-write` or `strong-read` message of sop_i. Thus, $w_k(x,u)$ is executed by p_i between the execution of $w_l(x,v)$ and the invocation of $r_i(x,v)$. This contradicts the assumption that $r_i(x,v)$ reads from $w_l(x,v)$. □

5.2.4. Obtaining the final sequences. Finally, from each process p_j, create a new sequence of operations τ_j by removing all delivery events from τ_j'''.

THEOREM 5.14. *Every execution generated by the algorithm is hybrid.*

Proof. For an execution σ, we have constructed a set of sequences $\{\tau_j\}_{j=1}^n$. We now show why this set satisfies the requirements of the definition of hybrid consistency.

Legality. By Lemma 5.13, every sequence τ_j is legal.

Condition 1. By construction, every sequence τ_j is a permutation of $ops(\sigma)$, which satisfies Condition 1 in the definition of hybrid consistency.

Condition 2. By Lemma 5.12, for every pair of operations op_l^1 and op_l^2 such that either op_l^1 or op_l^2 is strong, $op_l^1 \xrightarrow{\tau_j} op_l^2$ if and only if $op_l^1 \xrightarrow{\sigma} op_l^2$, which satisfies Condition 2 in the definition of hybrid consistency.

Condition 3. By Lemma 5.9, there exists a linearization ρ of all operations in $ops(\sigma)$ such that for every pair of strong operations sop_l and sop_k, $sop_l \xrightarrow{\tau_j} sop_k$ if and only if $sop_l \xrightarrow{\rho} sop_k$, which satisfies Condition 3 in the definition of hybrid consistency.

Condition 4. By Lemma 5.10, for every sequence τ_j, $\tau_j|j = \sigma|j$, which satisfies Condition 4 in the definition of hybrid consistency. □

5.3. Complexity analysis. Since the algorithm uses an atomic broadcast mechanism, the actual time and message complexity depends on the complexity of this

mechanism. We will make the complexity analysis with respect to t_{abc}, the time required to broadcast a message using **abcast**, t_{send}, the time required to send a message using **asend**, n_{abc}, the number of physical messages required to perform **abcast**, and n_{send}, the number of physical messages required to perform **asend**. We ignore the effects of contention on the network in this time analysis.

Weak reads are executed instantaneously from the local copy of the object they access and cause no messages to be sent. Weak writes are also executed instantaneously, but each weak write requires n_{abc} physical messages in order to broadcast its `update` message and $n \cdot n_{send}$ physical messages for the corresponding `ack` messages.

A strong operation must wait for all `ack` messages of previous weak write operations and `dummy` messages, which could take $t_{abc} + t_{send}$ time. Then, it broadcasts its own message and must wait until all `ack` messages are delivered, which could take an additional $t_{abc} + t_{send}$ time. The total time required to execute a strong operation is therefore $2(t_{abc} + t_{send})$.

A strong write operation requires n_{abc} physical messages in order to broadcast its `strong-write` message and $n \cdot n_{send}$ physical messages for the corresponding `ack` messages. The total number of physical messages caused by a strong write operation is therefore $n_{abc} + n \cdot n_{send}$.

A strong read operation requires n_{abc} physical messages in order to broadcast its `dummy` message and $n \cdot n_{send}$ physical messages for the corresponding `ack` messages if the last previous operation was a weak read. Then, an additional n_{abc} physical messages are required to broadcast its `strong-read` message plus $n \cdot n_{send}$ physical messages for the corresponding `ack` messages. The total number of physical messages caused by every strong read operation is therefore $2(n_{abc} + n \cdot n_{send})$.

Much work has been done on atomic broadcast, and several atomic broadcast algorithms have been developed, e.g., [6, 7, 8, 10, 12, 18, 19, 23, 25, 26, 27, 31, 43, 59]. Several of these protocols guarantee that messages are delivered within $2d$ time [10, 18, 26, 43]. Assuming a delivery time of $2d$, the time complexity of strong operations is $8d$, or in other words, linear with the network delay, which matches our goal.

The exact values of n_{abc} and n_{send} highly depend on the specific atomic broadcast mechanism being used and whether it uses hardware multicast or not. If we use a simple sequencer scheme, such as proposed in [10], then n_{abc} is 2 if we assume hardware multicast and n otherwise, while n_{send} is 2 in any case. By using these numbers in the above analysis, we get 0 messages for weak reads, $2 + 2n$ messages for weak or strong writes assuming hardware multicast and $3n$ otherwise, and $4 + 4n$ messages for strong reads assuming hardware multicast and $6n$ otherwise.

5.4. Relaxing the complete copy assumption. As done in other theoretical papers on this topic (e.g., [4, 5, 42]) our implementation assumes that each process maintains a copy of the entire memory. This assumption was made for clarity of presentation, but it can be relaxed as follows. The memory may be split among the processes, such that every memory object is held by at least one process. Whenever a memory operation is invoked at a process p_i, if p_i owns a copy of the object, then the code is executed as in the case where each member has a complete copy of the entire memory. If p_i does not hold a copy of that object, then it behaves in one of the following ways, depending on the type of the operation: if the operation is a weak or strong write, then p_i follows the same code as before, except for not updating the copy that it does not have. If the operation is a weak read, p_i sends a message to the nearest copy of the object using **asend**, inquiring about the value of this object, and returns the answer. Finally, if the operation is a strong read, p_i sends the `strong-read`

message as before, and one of the holders of the object returns the value of this object at the time it received the `strong-read` message. When p_i receives all the replies, it returns this value to the application program.

Note that there is a tradeoff here between memory efficiency and execution latencies: in order to always be able to execute weak reads instantaneously, every process must hold a complete copy of the entire memory; the more copies of each memory object exist, the more likely it is for a weak read to be executed immediately. This tradeoff exists in other consistency conditions as well.

6. Lower bounds for implementing hybrid consistency. It is known [12] that in any (asynchronous) implementation of linearizability, the response time of ("strong") read and write operations is $\Omega(d)$. Thus, our algorithm shows that it is possible to implement hybrid consistency in such a way that weak operations are extremely fast, without sacrificing the response time of strong operations.

Our implementation of hybrid consistency supports Definition 3.1, that is, hybrid consistency where the strong operations are linearizable. In contrast, sequential consistency can be implemented more efficiently than linearizability [12]. This might lead one to think that hybrid consistency where the strong operations are sequentially consistent can be implemented with a better asymptotic response time than our implementation. The following theorems show that this is not the case, at least if weak operations are required to be "fast." These theorems are a variation of a similar theorem which appears in [46]. The lower bounds proved in this section apply to both flavors of hybrid consistency. Recall that $|op|$ denotes the maximum time required to execute an operation of type op.

THEOREM 6.1. *For any implementation of hybrid consistency,* $|read| + |swrite| \geq d$.

Proof. Assume, by way of contradiction, that there is an implementation of hybrid consistency for which $|read| + |swrite| < d$. Let p_1 and p_2 be two processes that access x and y. Without loss of generality, assume that x and y are initially 0.

By the specification of y, there is a hybrid execution σ_1 such that $ops(\sigma_1)$ is

$$[\mathrm{SWrite}_1(x,1),\ \mathrm{SAck}_1(x)],\ [\mathrm{Read}_1(y),\ \mathrm{Return}_1(y,0)],$$

$\mathrm{SWrite}_1(x,1)$ occurs at real time 0, and $\mathrm{Read}_1(y)$ occurs immediately after $\mathrm{SAck}_1(x)$. The delay of all messages in σ_1 is exactly d. By assumption, the real time at the end of σ_1 is less than d. Hence, no message is received at any node during σ_1.

Similarly, by the specification of x, there is some hybrid execution σ_2 such that $ops(\sigma_2)$ is

$$[\mathrm{SWrite}_2(y,1),\ \mathrm{SAck}_2(y)],\ [\mathrm{Read}_2(x),\ \mathrm{Return}_2(x,0)],$$

$\mathrm{SWrite}_2(y,1)$ occurs at real time 0, and $\mathrm{Read}_2(x)$ occurs immediately after $\mathrm{SAck}_2(y)$. The delay of all messages in σ_2 is exactly d. By assumption, the real time at the end of σ_2 is less than d. Hence, no message is received at any node during σ_2.

Since no message is ever received in σ_1 and σ_2, the result of combining p_1's history in σ_1 with p_2's history in σ_2 can be extended to a hybrid execution σ. Then $ops(\sigma)$ consists of the operation $[\mathrm{SWrite}_1(x,1),\ \mathrm{SAck}_1(x)]$ followed by $[\mathrm{Read}_1(y),\ \mathrm{Return}_1(y,0)]$ and $[\mathrm{SWrite}_2(y,1),\ \mathrm{SAck}_2(y)]$ followed by $[\mathrm{Read}_2(x),\ \mathrm{Return}_2(x,0)]$.

Since σ is hybrid, there exists a serialization ρ of $ops(\sigma)$ such that for every $i \in \{1,2\}$, there exists a legal serialization τ_i of $ops(\sigma)$ that preserves: (a) the order of operations at p_i, (b) the order between strong and weak operations at all

processes, and (c) the order between strong operations in ρ (as formalized in Definition 3.2). Without loss of generality, assume that in ρ, [SWrite$_1(x,1)$, SAck$_1(x)$] precedes [SWrite$_2(y,1)$, SAck$_2(y)$].

Consider τ_2. Since τ_2 is legal, each read should precede the strong write to the same variable (by the other process) in τ_2. Thus, in τ_2, [Read$_2(x)$, Return$_2(x,0)$] precedes [SWrite$_1(x,1)$, SAck$_1(x)$], which, in turn, precedes [SWrite$_2(y,1)$, SAck$_2(y)$]. Therefore, in τ_2, [Read$_2(x)$, Return$_2(x,0)$] precedes [SWrite$_2(y,1)$, SAck$_2(y)$], but then $\tau_2 \mid 2 \neq \sigma \mid 2$. This is a contradiction. □

The interested reader can compare this proof with the proof of the similar result in [12, Theorem 3.1] to see the extra care needed when arguing about properties of hybrid executions.

Using the same arguments as in the proof of Theorem 6.1, only reversing the roles of reads and writes, we can prove the following theorem.

THEOREM 6.2. *For any implementation of hybrid consistency,* $|write| + |sread| \geq d$.

We define a *fast write implementation* to be one in which the time required to execute weak write operations is strictly less than $d/2$. Likewise, a *fast read implementation* is an implementation in which the time required to execute weak read operations is strictly less than $d/2$. Note that the implementation given in the previous section is fast—weak operations return instantaneously. We immediately obtain the following corollary.

COROLLARY 6.3. *Let I be a fast read implementation of hybrid consistency. Then* $|swrite| \geq d/2$.

COROLLARY 6.4. *Let I be a fast write implementation of hybrid consistency. Then* $|sread| \geq d/2$.

That is, in any implementation of hybrid consistency in which weak reads are executed faster than the network delay, strong writes must be slower than the network delay, and if weak writes are faster than the network delay, then strong reads must be slower than the network delay. In that sense, the implementation presented in this paper is optimal (up to a constant factor).

7. Discussion and further research. This paper presents a theoretical study of hybrid consistency. This study is motivated by several new processors and multiprocessors that support some sort of weak and strong operations; it is also motivated by recent theoretical results on the cost of supporting global consistency conditions and the inadequacy of weak consistency conditions. We have presented a formal and precise definition of hybrid consistency in two flavors, one based on sequential consistency and the other based on linearizability. We have presented an algorithm for providing hybrid consistency, proved its correctness, and analyzed its performance. We have also shown lower bounds on the response time of any implementation of hybrid consistency that does not delay weak operations (which are within constant factors of the bounds achieved by our algorithm).

Our definition of hybrid consistency (Definition 3.2) assumes that operations of the same process are executed one at a time and in program order. Recently, our definition has been generalized to allow nonsequential executions [9], i.e., executions in which operations may be executed in parallel, in pipeline, and even out of order. The definition in [9] also incorporates a formal treatment of control operations, which is not part of our definition. Naturally, the result is a slightly more complicated definition. We feel that Definition 3.2 is still interesting since it sheds light on hybrid

consistency: how to program with it, what its costs are, and possible algorithms for providing it. Furthermore, to the best of our knowledge, there is no formally verified algorithm for providing hybrid consistency when operations can be issued in parallel, in pipeline, and out of order.

Our work leaves open several theoretical, as well as practical, questions.

We would like to obtain tighter bounds (for time, messages, and congestion) for implementations of hybrid consistency. In this work we assumed that local computations are negligible and that messages can be sent to the network without any limitations. It is important to obtain lower and upper bounds on implementations of hybrid consistency and other consistency conditions in a more realistic model. The Postal [13] or the LogP [28] models might serve as a middle ground between our theoretical assumptions and realistic architectures.

It would be interesting to quantify what the performance benefits of programming with hybrid consistency are. Specifically, are algorithms for solving a specific problem that rely on hybrid consistency faster than algorithms for solving the same problem that rely on other consistency conditions? If so, how much? These questions can be answered either by theoretical or experimental methods.

In this paper, we have made a small step towards developing proof techniques for hybrid consistency by showing how to argue about the correctness of an example program. Subsequent work [9] presents several techniques for transforming programs written for sequentially consistent hardware into correct and efficient programs for hybrid consistent hardware; related results for alpha consistency appear in [11].

Another interesting and important issue is the design of compiler techniques for using hybrid consistency. Such a compiler can, for example, take a program which was written assuming a sequentially consistent or linearizable memory, and decide automatically for each memory access operation whether it should be weak or strong. This should be done without affecting the correctness of the program and while achieving the maximal possible speedup. (Such techniques for sequentially consistent memories that allow pipelining appear in [56].)

The problem of verifying whether a given execution is sequentially consistent was investigated by Gibbons and Korach [36]. They have shown that for most interesting cases, this problem is NP-complete, although under some restrictions it can be solved in polynomial time. It is clear that the problem of verifying whether a given execution is hybrid consistent is at least as hard. However, it is unclear whether verifying hybrid consistency is harder than verifying sequential consistency.

The ultimate goal of our research is to obtain a better understanding of memory consistency conditions, so it would be possible to evaluate several conditions and decide which is best suited for a specific application and architecture. In order to do so, a wide spectrum of properties needs to be considered, such as performance, computation power, and complexity of the possible implementations. It is necessary to formalize these properties and develop criteria for evaluating consistency conditions.

Acknowledgments. Jennifer Welch collaborated with us in the early stages of this research. We are indebted to her for her contributions to the definition of hybrid consistency and for many helpful discussions and comments on several versions of this paper. Thanks also to Sarita Adve, Eran Aharonson, Ken Birman, Kourosh Gharachorloo, Maurice Herlihy, Martha Kosa, Rivka Ladin, Friedemann Mattern, Lihu Rappoport, and Tali Shintel for helpful remarks.

REFERENCES

[1] S. Adve and M. Hill, *Weak ordering—A new definition*, in Proc. 17th ACM/IEEE International Symp. on Computer Architecture, 1990, pp. 2–14.

[2] S. Adve and M. Hill, *A Unified Formalization of Four Shared-Memory Models*, Technical Report 1051, Computer Science Department, University of Wisconsin, Madison, 1991.

[3] S. Adve and M. Hill, *Sufficient Conditions for Implementing the Data-Race-Free-*1 *Memory Model*, Technical Report 1107, Computer Science Department, University of Wisconsin, Madison, 1992.

[4] M. Ahamad, P. Hutto, and R. John, *Implementing and Programming Causal Distributed Shared Memory*, Technical Report TR GIT-CC-90-49, Georgia Institute of Technology, Atlanta, 1990.

[5] M. Ahamad, G. Neiger, P. Kohli, J. Burns, and P. Hutto, *Causal memory: Definitions, implementation, and programming*, Distrib. Computing, 9 (1995), pp. 37–49.

[6] G. Alvarez, F. Cristian, and S. Mishra, *On-Demand Asynchronous Atomic Broadcast*, Technical Report CS95-416, Dept. of Computer Science, University of California, San Diego, 1995.

[7] Y. Amir, D. Dolev, S. Kramer, and D. Malki, *Total Ordering of Messages in Broadcast Domains*, Technical Report CS92-9, Dept. of Computer Science, The Hebrew University of Jerusalem, 1992.

[8] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D.A. Agarwal, and P. Ciarfella, *Fast message ordering and membership using a logical token-passing ring*, in Proc. 13th IEEE International Conf. on Distributed Computing Systems, 1993, pp. 551–560.

[9] H. Attiya, S. Chaudhuri, R. Friedman, and J. Welch, *Shared memory consistency conditions for non-sequential execution: Definitions and programming strategies*, SIAM J. Comput., 27 (1998), to appear.

[10] H. Attiya and R. Friedman, *A Correctness Condition for High-Performance Multiprocessors*. Technical Report #767, Department of Computer Science, Technion, Haifa, Israel, 1993.

[11] H. Attiya and R. Friedman, *Programming DEC-alpha based multiprocessors the easy way*, in Proc. 6th ACM Symp. on Parallel Algorithms and Architectures, ACM, New York, 1994, pp. 157–166; also, Technical Report LPCR #9411, Department of Computer Science, Technion, Haifa, Israel.

[12] H. Attiya and J. Welch, *Sequential consistency versus linearizability*, ACM Trans. Comput. Systems, 12 (1994), pp. 91–122.

[13] A. Bar-Noy and S. Kipnis, *Designing broadcasting algorithms in the postal model for message-passing systems*, in Proc. 4th ACM Symp. on Parallel Algorithms and Architectures, ACM, New York, 1992, pp. 11–22.

[14] J. Bennett, J. Carter, and W. Zwaenepoel, *Munin: Distributed shared memory based on type-specific memory coherence*, in Proc. 2nd ACM Symp. on Principles and Practice of Parallel Processing, ACM, New York, 1990, pp. 168–176.

[15] P. Bernstein, V. Hadzilacos, and H. Goodman, *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, MA, 1987.

[16] K. Birman, *The process group approach to reliable distributed computing*, Comm. ACM, 36 (1993), pp. 37–53.

[17] K. Birman and T. Joseph, *Exploiting virtual synchrony in distributed systems*, in Proc. 11th ACM Symp. on Operating Systems Principles, ACM, New York, 1987, pp. 123–138.

[18] K. Birman and T. Joseph, *Reliable communication in the presence of failures*, ACM Trans. Comput. Systems, 5 (1987), pp. 47–76.

[19] K. Birman, A. Schiper, and P. Stephenson, *Lightweight causal and atomic group multicast*, ACM Trans. Comput. Systems, 9 (1991), pp. 272–314.

[20] R. Bisiani, A. Nowatzyk, and M. Ravishankar, *Coherent Shared Memory on a Distributed Memory Machine*, in Proc. International Conf. on Parallel Processing, Pennsylvania State University, University Park, PA, 1989, pp. I–133–141.

[21] W. Brantley, K. McAuliffe, and J. Weiss, *RP*3 *processor-memory element*, in Proc. International Conf. on Parallel Processing, Pennsylvania State University, University Park, PA, 1985, pp. 782–789.

[22] L. M. Censier and P. Feautrier, *A new solution to coherence problems in multicache systems*, IEEE Trans. Comput., C-27 (1978), pp. 1112–1118.

[23] J. Chang and N. Maxemchuk, *Reliable broadcast protocols*, ACM Trans. Comput. Systems, 2 (1984), pp. 251–273.

[24] W. W. Collier, *Architectures for Systems of Parallel Processes*, Technical Report IBM TR 00.3253, IBM, Poughkeepsie, NY, 1984.

[25] F. Cristian, *Asynchronous Atomic Broadcast*, IBM Technical Disclosure Bulletin, 33, 1991, pp. 115–116; also, 1st IEEE Workshop on Management of Replicated Data, Houston, TX, IEEE, Piscataway, NJ, 1990.

[26] F. Cristian, R. Beijer, and S. Mishra, *A performance comparison of asynchronous atomic broadcast protocols*, Distrib. Systems Engrg. J., 1 (1994), pp. 177–201.

[27] F. Cristian and S. Mishra, *The pinwheel asynchronous atomic broadcast protocols*, in Proc. 2nd International Symp. on Autonomous Decentralized Systems, Phoenix, AZ, 1995; also, Technical Report CSE93-331, Department of Computer Science & Engineering, University of California, San Diego.

[28] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eiken, *LogP: Towards a realistic model of parallel computation*, in ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM, New York, 1993. Also: Technical Report UCB/CS/92 713, University of California, Berkeley.

[29] M. Dubois and C. Scheurich, *Memory access dependencies in shared-memory multiprocessors*, IEEE Trans. Software Engrg., 16 (1990), pp. 660–673.

[30] M. Dubois, C. Scheurich, and F. A. Briggs, *Synchronization, coherence and event ordering in multiprocessors*, IEEE Trans. Comput., 21 (1988), pp. 9–21.

[31] P. Ezhilchelvan, R. Macedo, and S. Shrivastava, *Newtop: A Fault-Tolerant Group Communication Protocol*, Technical Report, Computer Science Department, University of Newcastle, Newcastle upon Tyne, UK, 1994.

[32] R. Friedman, *Consistency Conditions for Distributed Shared Memories*. Ph.D. Thesis, Department of Computer Science, Technion, Haifa, Israel, 1994.

[33] R. Friedman, *Implementing hybrid consistency with high-level synchronization operations*, Distrib. Comput., 9 (1995), pp. 119–129.

[34] K. Gharachorloo, A. Gupta, and J. Hennessy, *Performance evaluation of memory consistency models for shared-memory multiprocessors*, in Proc. 4th ACM International Conf. on Architectural Support for Programming Languages and Operating Systems, ACM, New York, 1991, pp. 245–257.

[35] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy, *Memory consistency and event ordering in scalable shared-memory multiprocessors*, in Proc. 17th ACM/IEEE International Symp. on Computer Architecture, 1990, pp. 15–26.

[36] P. Gibbons and E. Korach, *Testing shared memories*, SIAM J. Comput., 26 (1997), pp. 1208–1244.

[37] P. Gibbons and M. Merritt, *Specifying non-blocking shared memories*, in Proc. 4th ACM Symp. on Parallel Algorithms and Architectures, ACM, New York, 1992, pp. 306–315.

[38] P. Gibbons, M. Merritt, and K. Gharachorloo, *Proving sequential consistency of high-performance shared memories*, in Proc. 3rd ACM Symp. on Parallel Algorithms and Architectures, ACM, New York, 1991, pp. 292–303.

[39] R. J. Goodman, *Cache Consistency and Sequential Consistency*, Technical Report 1006, Computer Science Department, University of Wisconsin, Madison, 1991.

[40] A. Gupta, J. Hennessy, K. Gharachorloo, T. Mowry, and W. Weber, *Comparative evaluation of latency reducing and tolerating techniques*, in Proc. 18th ACM/IEEE International Symp. on Computer Architecture, May 1991, pp. 254–263.

[41] M. Herlihy and J. Wing, *Linearizability: A correctness condition for concurrent objects*, ACM Trans. Programming Languages Systems, 12 (1990), pp. 463–492.

[42] J. James and A. Singh, *Complete implementations for shared memory consistency conditions*, in Proc. 14th ACM Symp. on Principles of Distributed Computing, ACM, New York, 1995, p. 273.

[43] F. Kaashoek, A. Tanenbaum, S. Hummel, and H. Bal, *An efficient reliable broadcast protocol*, Operating Systems Review, 23 (1989), pp. 5–19.

[44] L. Lamport, *How to make a multiprocessor computer that correctly executes multiprocess programs*, IEEE Trans. Comput., C-28 (1979), pp. 690–691.

[45] K. Li and P. Hudak, *Memory coherence in shared virtual memory systems*, ACM Trans. Comput. Systems, 7 (1989), pp. 321–359.

[46] R. Lipton and J. Sandberg, *PRAM: A Scalable Shared Memory*, Technical Report CS-TR-180-88, Computer Science Department, Princeton University, Princeton, NJ, 1988.

[47] K. Marzullo, R. Cooper, M. Wood, and K. Birman, *Tools for distributed applications management*, IEEE Trans. Comput., 24 (1991), pp. 42–51.

[48] S. Min and J. Baer, *A timestamp-based cache coherence scheme*, in Proc. International Conf. on Parallel Processing, Pennsylvania State University, University Park, PA, 1989, pp. I–23–32.

[49] C. Papadimitriou, *The Theory of Concurrency Control*, Computer Science Press, Rockville, MD, 1986.
[50] G. L. Peterson, *Myths about the mutual exclusion problem*, Inform. Process. Lett., 12 (1981), pp. 115–116.
[51] U. Ramachandran, M. Ahamad, and M. Y. Khalidi, *Coherence of distributed shared memory: Unifying synchronization and data transfer*, in Proc. International Conf. on Parallel Processing, Pennsylvania State University, University Park, PA 1989, pp. II–160–169.
[52] M. Raynal, *Algorithms for Mutual Exclusion*, MIT Press, Cambridge, MA, 1986.
[53] P. Reynolds, C. Williams, and R. Wagner, *Empirical Analysis of Isotach Networks*, Technical Report 92-19, Dept. of Computer Science, University of Virginia, Richmond, 1992.
[54] C. Scheurich and M. Dubois, *Correct memory operation of cache-based multiprocessors*, in Proc. 14th ACM/IEEE International Symp. on Computer Architecture, 1987, pp. 234–243.
[55] A. Setubal and G. Gerber, *Fault-tolerant distributed database for supervisory control system*, in Proc. Workshop Verteilte Datenbaksysteme, Karlsruhe University, Karlsruhe, Germany, 1991.
[56] D. Shasha and M. Snir, *Correct and efficient execution of parallel programs that share memory*, ACM Trans. Programming Languages Systems, 10 (1988), pp. 282–312.
[57] A. Siegel, K. Birman, and K. Marzullo, *Deceit: A Flexible Distributed File System*, Technical Report TR 89-1042, Department of Computer Science, Cornell University, Ithaca, NY, 1989.
[58] R. van Renesse, K. Birman, R. Friedman, M. Hayden, and D. Karr, *A framework for protocol composition in horus*, in Proc. 14th ACM Symposium on Principles of Distributed Computing, ACM, New York, 1995, pp. 80–89.
[59] R. van Renesse, K. Birman, and S. Maffeis, *Horus: A flexible group communication system*, Comm. ACM, 39 (1996), pp. 76–83.
[60] R. N. Zucker and J.-L. Baer, *A Performance Study of Memory Consistency Models*, in Proc. 19th ACM/IEEE International Symp. on Computer Architecture, 1992, pp. 2–12.

SIAM J. COMPUT.
Vol. 27, No. 6, pp. 1671–1694, December 1998
© 1998 Society for Industrial and Applied Mathematics
008

EFFICIENT ALGORITHMS FOR THE DOMINATION PROBLEMS ON INTERVAL AND CIRCULAR-ARC GRAPHS*

MAW-SHANG CHANG†

Abstract. This paper first presents a unified approach to design efficient algorithms for the weighted domination problem and its three variants, i.e., the weighted independent, connected, and total domination problems, on interval graphs. Given an interval model with endpoints sorted, these algorithms run in time $O(n)$ or $O(n \log \log n)$ where n is the number of vertices. The results are then extended to solve the same problems on circular-arc graphs in $O(n+m)$ time where m is the number of edges of the input graph.

Key words. interval graphs, circular-arc graphs, domination, graph algorithms

AMS subject classifications. 05C85, 68Q25, 68Q20, 68R10, 90C27

PII. S0097539792238431

1. Introduction.

Let $G = (V, E)$ be an undirected graph. Throughout the paper, n and m denote the numbers of vertices and edges of a graph, respectively. For a vertex $v \in V$, let $N(v) = \{u : u \in V,\ u \neq v$ and u is a neighbor of $v\}$ and $N[v] = N(v) \cup \{v\}$. In general, for $S \subseteq V$, $N(S)$ and $N[S]$ denote $\cup_{v \in S} N(v)$ and $N(S) \cup S$, respectively. A subset S of V dominates another subset T of V if $T \subseteq N[S]$. A dominating set S of a subset T of V is called an *independent, connected, and total dominating set* of T if the induced subgraph of S has no edges, is connected, and has no isolated vertices, respectively. A subset S of V is called an (*independent, connected, total*) *dominating set* of graph G if S is an (*independent, connected, total*) *dominating set* of V. Let each vertex of graph G be associated with a real weight. The weight of a set of vertices is the sum of the weights of all vertices in the set. For a vertex v and a set S of vertices, $w(v)$ and $w(S)$ denote the weights of v and S, respectively. The weighted domination problem involves finding a dominating set of minimum weight for a weighted graph. The weighted independent (connected, total) domination problem involves finding an independent (connected, total) dominating set of minimum weight for a weighted graph. Given a graph G and an integer K, the problem of determining whether G has a dominating set whose cardinality is less than K is NP-complete [17]. The same holds true for independent, connected, and total domination [17], [27].

A graph $G = (V, E)$ is called an *intersection graph* for a finite family $\mathcal{F}$ of a nonempty set if there is a one-to-one correspondence between $\mathcal{F}$ and V such that two sets in $\mathcal{F}$ have nonempty intersection if and only if their corresponding vertices in V are adjacent to each other. We call $\mathcal{F}$ an *intersection model* of G. For an intersection model $\mathcal{F}$, we use $G(\mathcal{F})$ to denote the intersection graph for $\mathcal{F}$. If $\mathcal{F}$ is a family of intervals on a real line, then G is called an *interval graph* for $\mathcal{F}$ and $\mathcal{F}$ is called an *interval model* of G. If $\mathcal{F}$ is a family of arcs on a circle, then G is called a *circular-arc graph* for $\mathcal{F}$ and $\mathcal{F}$ is called a *circular-arc model* of G.

* Received by the editors October 13, 1992; accepted for publication (in revised form) September 9, 1996; published electronically JUne 3, 1998. This research was supported in part by the National Science Council of the Republic of China under grant NSC 81-0408-E-194-04. An extended abstract of this work appeared in the *Proceedings of the* 12*th IFIP World Computer Congress*.
http://www.siam.org/journals/sicomp/27-6/23843.html

† Department of Computer Science and Information Engineering, National Chung Cheng University, Min-Hsiung, Chiayi 621, Taiwan, Republic of China (mschang@cs.ccu.edu.tw).

Booth and Lueker [10] gave an $O(n+m)$-time algorithm for recognizing an interval graph and constructing, in the affirmative case, an interval model using PQ-trees. Also, the endpoints of constructed intervals can be restricted to be distinct integers between 1 and $2n$ [31]. We refer to a set of intervals satisfying that the endpoints are distinct integers between 1 and $2n$ as a set of *sorted intervals* since they can be sorted easily in $O(n)$ time. Korte and Möhring [26] simplified the operations on a PQ-tree. Ramalingan and Pandu Rangan gave a simple $O(n^2)$ time algorithm which can be implemented in parallel easily [32]. Simon [35], Hsu and Ma [24] gave new algorithms without using PQ-trees. Interval graphs have been used in many practical applications [19]. If we are given an interval model whose endpoints are not integers between 1 and $2n$, we can sort the endpoints in nondecreasing order and use the indices of endpoints in the sorted list to construct a new interval model whose endpoints are distinct integers between 1 and $2n$. Thus, we refer to a set of intervals satisfying that the endpoints are distinct integers between 1 and $2n$ as a set of *sorted intervals*.

Tucker [39] proposed an $O(n^3)$-time algorithm for recognizing a circular-arc graph and constructing, in the affirmative case, a circular-arc model. Hsu gave an $O(nm)$-time algorithm [23] and Eschen and Spinrad presented an $O(n^2)$-time algorithm [13]. Circular-arc graphs have a variety of applications involving traffic light sequencing, genetics, and others. Circular-arc graphs have been studied in [18], [19], [22], [9]. Similarly, we refer to a set of arcs satisfying that the endpoints are distinct integers between 1 and $2n$ as a set of *sorted arcs*.

Investigating the algorithmic complexity of total domination on interval graphs was mentioned in [27] as a relevant open question. An $O(n^2)$-time algorithm has been proposed for this problem in [3]. Later, Ramalingan and Pandu Rangan gave an $O(n+m)$ algorithm [30], and Bertossi and Gori [5] presented an $O(n \log n)$-time algorithm for this problem. Ramalingam and Pandu Rangan [31] presented a unified approach to solve the weighted versions of various domination problems mentioned above on interval graphs in $O(m+n)$ time. Bonuccelli [9] gave an $O(nm)$-time algorithm for finding a minimum-cardinality dominating set on circular-arc graphs. Bertossi and Moretti [6] presented $O(n^3)$ algorithms for finding a minimum-weight dominating set and a minimum-weight total dominating set on circular-arc graphs. Recently, an $O(n)$-time algorithm for finding a minimum-cardinality dominating set of a circular-arc graph given a set of sorted arcs was given by Hsu and Tsai [25] and an $O(n^2 \log n)$ algorithm for the weighted version was given by Asano [2]. Interval graphs and circular-arc graphs have been studied by many researchers [20]. We only mention some results related to the domination problem studied in this paper. For computing the independence number of circular-arc graphs, see [21], [29]. For the irredundance problem on circular-arc graphs, please refer to [11]. For parallel algorithms for various domination problems and some other problems in interval graphs and circular-arc graphs, we refer the reader to [4], [6], [33], [36], [40].

In this paper, we assume that weights are real for the independent domination problem and assume that weights are nonnegative for the domination, connected, and total domination problems. Manacher and Mankus [28] showed that any algorithm for finding a minimum-weight dominating set, connected dominating set, and a total dominating set of a graph with nonnegative weights can be extended to incorporate negative-weight vertices without loss of efficiency. We will present a new unified approach to solve the weighted domination, independent, connected, and total domination problems on interval graphs. Given a set I of sorted intervals, the algorithms solve these problems on $G(I)$ in $O(n)$ or $O(n \log\log n)$ time and $O(n)$ space. These

algorithms are straightforward and simple but efficient. Then we extend the results to solve the same four problems on circular-arc graphs given sorted arc models in $O(m+n)$ time and $O(n)$ space. The results of this paper are summarized in section 4.

2. Algorithms for various domination problems on interval graphs. In this section, we study algorithms for various domination problems [8, 12, 20] on interval graphs. It is assumed that the input graph is given by an interval model I which is a set of n sorted intervals labeled by $1, 2, \ldots, n$ in increasing order of their right endpoints. The left endpoint of interval i is denoted by a_i and the right endpoint by b_i. We say that interval i starts from endpoint a_i and finishes at endpoint b_i. By definition, $1 \leq a_i < b_i \leq 2n$ for $1 \leq i \leq n$. For convenience, we need the following notation.

(i) For a set S of intervals, the largest (respectively, smallest) right endpoint of intervals in S is denoted by $\max b(S)$ (respectively, $\min b(S)$); the interval in S with the largest (respectively, smallest) right endpoint is denoted by $last(S)$ (respectively, $first(S)$); the largest left endpoint of intervals in S is denoted by $\max a(S)$. For technical reasons, we let $\max a(S) = 0$ if S is empty.

(ii) For endpoint e, we use $interval(e)$ to denote the interval whose left or right endpoint is equivalent to e, $IFB(e)$ (intervals finishing before endpoint e) to denote the set of all intervals whose right endpoints are less than e, and $ISB(e)$ (intervals starting before endpoint e) to denote the set of all intervals whose left endpoints are less than e. Thus, $\max a(IFB(e))$ is the largest left endpoint of the intervals whose right endpoints are less than e. Clearly, $interval(\max a(S))$ is the interval with the largest left endpoint in S.

(iii) A subset S of I is called a *partial dominating* (PD) set if S dominates all intervals starting before the right endpoint of $last(S)$. A PD set S is called an *independent partial dominating* (IPD) set if S is an independent set. A PD set S is called a *connected partial dominating* (CPD) set if the subgraph $G(S)$ induced by S is connected. A PD set S is a *total partial dominating* (TPD) set if the subgraph $G(S)$ induced by S has no isolated vertices.

(iv) For a family X of sets of vertices, $\mathrm{Min}(X)$ denotes a minimum-weight vertex set in X if X is not the empty set; $\mathrm{Min}(X)$ denotes a set of infinite weight otherwise.

(v) Suppose L is a list of numbers. We use δ_L and τ_L to denote the head and tail of L, respectively. For number j, we refer to the smallest number in L which is larger than j, denoted by $succ_L(j)$, as the successor of j in L and the largest number in L which is less than j, denoted by $pred_L(j)$, as the predecessor of j in L, respectively. For simplicity, we use δ and τ, $succ(j)$, and $pred(j)$ when L is understood without ambiguity.

To readers familiar with notation, we give a set I of seven intervals shown in Figure 1 for illustration. For example, $ISB(b_1) = \{1, 2, 4\}$, $ISB(b_3) = \{1, 2, 3, 4\}$, $ISB(b_5) = I$, $interval(5) = 3$, and $interval(10) = 4$. For example, $IFB(a_1) = \emptyset$, $IFB(a_4) = \emptyset$, $IFB(a_3) = \{1\}$, $IFB(b_2) = \{1\}$, and $IFB(a_6) = \{1, 2, 3, 4\}$. It is easy to verify that $\max a(IFB(a_1)) = 0$, $\max a(IFB(a_4)) = 0$, $\max a(IFB(a_3)) = a_1$, $\max a(IFB(b_2)) = a_1$, and $\max a(IFB(a_6)) = a_3$. For $S = \{1, 2, 3, 4\}$, we have that $\min b(S) = 4$, $\max b(S) = 10$, $\max a(S) = 5$, $first(S) = 1$, $last(S) = 4$, and $interval(\max a(S)) = 3$. Since $\{1, 3\}$ dominates $ISB(b_3)$, it is a PD set of $G(I)$. Clearly, it is also an IPD set. It is easy to see that $\{4, 5\}$ is a connected and a TPD set. Suppose that $w(1) = 11$, $w(2) = 5$, $w(3) = 1$, $w(4) = 2.5$, $w(5) = 7$, $w(6) = 6$, and $w(7) = 3.4$. For $X = \{\{1, 3, 7\}, \{2, 5\}, \{2, 4, 6\}, \{1, 3\}\}$, which is a family of

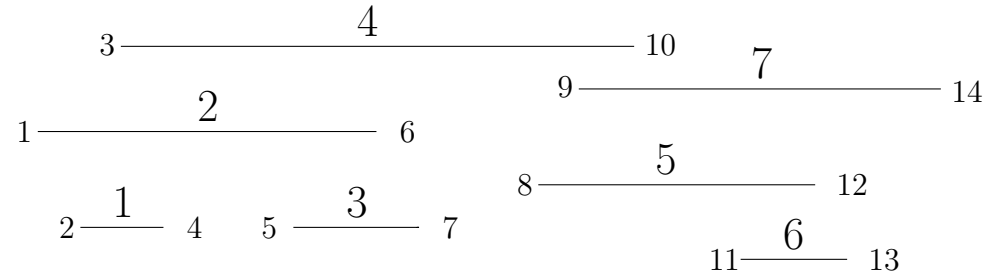


FIG. 1. *A family of seven intervals.*

four subsets of I, $\text{Min}(X) = \{2,5\}$ or $\text{Min}(X) = \{1,3\}$ since $w(\{1,3,7\}) = 15.4$, $w(\{2,5\}) = 12$, $w(\{2,4,6\}) = 13.7$, and $w(\{1,3\}) = 12$.

Following the above definitions, we can prove the following two lemmas easily.

LEMMA 2.1. *For two endpoints e_1 and e_2 where $e_1 < e_2$ of intervals in I, there does not exist any interval $j \in I$ such that $e_1 < a_j < b_j < e_2$ if and only if $e_1 \geq \max a(IFB(e_2))$.*

Proof. The lemma is proved by definition. □

LEMMA 2.2. *For two intervals k and i of I where $k < i$, interval i dominates $ISB(b_i) - ISB(b_k)$ if and only if $b_k > \max a(IFB(a_i))$.*

Proof. It is easy to see that $ISB(b_i) - ISB(b_k)$ can be partitioned into two subsets Z_1 and Z_2, where $Z_1 \subseteq N[i]$ and $Z_2 = \{j : j \in I, b_k < a_j < b_j < a_i\}$. By Lemma 2.1, $Z_2 = \emptyset$ if and only if $b_k > \max a(IFB(a_i))$. Thus, interval i dominates $ISB(b_i) - ISB(b_k)$ if and only if $b_k > \max a(IFB(a_i))$. □

In the following subsections, we will give recursive formulas for various weighted domination problem in interval graphs. Although Ramalingan and Pandu Rangan also gave recursive formulas in [31], our recursive formulas are different from theirs.

2.1. Independent domination. Let I_d be the set of intervals obtained by augmenting I with two intervals 0 and $n+1$ with $w(0) = w(n+1) = 0$. The left and right endpoints of interval 0 are -1 and 0, respectively, and the left and right endpoints of interval $n+1$ are $2n+1$ and $2n+2$, respectively. That is, $I_d = I \cup \{0, n+1\}$. Then a subset S of I is an independent dominating set of $G(I)$ if and only if $S \cup \{0, n+1\}$ is an independent dominating set of $G(I_d)$. For technical reason, we solve the weighted independent domination problem on $G(I)$ by finding a minimum-weight independent dominating set of $G(I_d)$. In this subsection, the set of intervals considered is I_d if not specified explicitly. It is easy to see that all IPDs contain interval 0. Let $IPD(i)$ be the collection of all IPDs whose last intervals are interval i. Then a subset S of I_d is an independent dominating set of $G(I_d)$ if and only if $S \in IPD(n+1)$. For simplicity, we use $MIPD(i)$ to denote $\text{Min}(IPD(i))$. Then we have the following lemma.

LEMMA 2.3. *The following three statements are true:*

(1) $MIPD(0) = \{0\}$.

(2) *For $1 \leq i \leq n+1$, we have that*
$MIPD(i) = \{i\} \cup \text{Min}(\{MIPD(j) : \max a(IFB(a_i)) < b_j < a_i\})$.

(3) $MIPD(n+1)$ *is a minimum-weight independent dominating set of $G(I_d)$.*

Proof. It is easy to verify the correctness of statements (1) and (3). In the following, we prove the correctness of statement (2) by showing that (i) if $S \in IPD(k)$ where $\max a(IFB(a_i)) < b_k < a_i$, then $S \cup \{i\} \in IPD(i)$; and (ii) if $S \in IPD(i)$ and $k = last(S - \{i\})$, then $S - \{i\} \in IPD(k)$ and $\max a(IFB(a_i)) < b_k < a_i$.

First, we prove statement (i). By definition, $last(S) = k$, and S dominates $ISB(b_k)$. Since $b_k < a_i$ and S is an independent set, $S \cup \{i\}$ is also an independent set. By Lemma 2.2, interval i dominates $ISB(b_i) - ISB(b_k)$ since $\max a(IFB(a_i)) < b_k$. Thus, $S \cup \{i\}$ dominates $ISB(b_i)$. By definition, $S \cup \{i\} \in IPD(i)$.

Next, we prove statement (ii). Since S is an independent set, interval i is not adjacent to interval k where $k = last(S - \{i\})$. Thus, $b_k < a_i$. Since S dominates $ISB(b_i)$ and $b_k < a_i$, we have that $S - \{i\}$ dominates $ISB(b_k)$ and interval i dominates $ISB(b_i) - ISB(b_k)$. By definition, $S - \{i\} \in IPD(k)$. By Lemma 2.2, we have that $\max a(IFB(a_i)) < b_k$. □

The recursive relation in Lemma 2.3 can be used to compute a minimum-weight independent domination set. The algorithm scans the endpoints of I_d from left to right. It uses a list L to store some intervals in increasing order. For simplicity, let $WI(k) = w(MIPD(k))$.

ALGORITHM MIDS. Find a minimum-weight independent dominating set of $G(I_d)$ given a set I_d of sorted intervals.

Input. A set I_d of sorted intervals and the weights of all intervals.

Output. A minimum-weight independent dominating set of $G(I_d)$.

Method.

1. $L \leftarrow \{0\}$; $WI(0) \leftarrow 0$; $MIPD(0) \leftarrow \{0\}$;
2. **for** $e = 1$ **to** $2n + 1$ **do**
3. **If** e is the left endpoint of some interval i, i.e., $e = a_i$, **then**
4. $MIPD(i) = \{i\} \cup MIPD(\delta)$;
   The union operation is implemented by setting a pointer from i to δ.
5. $WI(i) = w(i) + WI(\delta)$ where δ is the head of L;
6. **else** e is the right endpoint of some interval i, i.e., $e = b_i$, **do**
7. Delete all elements of L whose right endpoints are less than a_i from the head of the list one by one;
8. **while** $L \neq \emptyset$ and $WI(\tau) > WI(i)$ **do**
9. Delete τ from L where τ is the tail of L;
   end while
10. Append i to the end of L;
    end if else
    end for
11. Output $MIPD(n + 1)$;

By the for loop (lines 2 to 10), the algorithm visits the endpoints of $I_d - \{0\}$ in increasing order one by one and maintains an invariant (invariant 1): right before the left endpoint of interval i to be visited, $MIPD(\delta) = \text{Min}(\{MIPD(j) : \max a(IFB(a_i)) < b_j < a_i\})$. Thus by statement (2) of Lemma 2.3, the results of lines 4 and 5 are correct. In the following, we show how the invariant is maintained true. In fact, the algorithm maintains some other invariants: right before endpoint e is to be visited, invariant 2, $L \subseteq \{j : j \in I, \max a(IFB(e)) < b_j < e\}$; invariant 3, the intervals in L are in increasing order of their right endpoints; and invariant 4, for f, $h \in L$, if $f < h$, then $WI(f) \leq WI(h)$. Apparently, all of these invariants are true initially. Invariant 2 holds because of the following two reasons: (i) the intervals whose right endpoints are less than the left endpoint of $interval(e)$ are removed from L when right endpoint e is being visited (line 7), and (ii) an interval is appended to the tail of L when its right endpoint is being visited (line 10). The intervals in L are easily maintained in increasing order of their right endpoints because (i) endpoints are visited in increasing order, and (ii) an interval is appended to the tail of L when

its right endpoint is being visited (line 10). Thus, invariant 3 holds. Invariant 4 holds since all intervals j with $WI(j) > WI(interval(e))$ are removed from L by the while loop (lines 8 and 9) when right endpoint e is being visited and $interval(e)$ is appended to the end of L (line 10). By invariants 3 and 4, $WI(\delta)$ is smallest among all $WI(j)$'s of $j \in L$. It is straightforward to verify that when the left endpoint of interval i is being visited, $\{j : j \in I, \max a(IFB(a_i)) < b_j < a_i\} - L$ are those intervals removed from L by the while loop (lines 8 and 9). Hence there exists a right endpoint in L, i.e., τ, such that $WI(j) > WI(\tau)$ for all $j \in \{j : j \in I, \max a(IFB(a_i)) < b_j < a_i\} - L$ when the left endpoint of interval i is being visited. By invariants 2, 3, 4, $WI(\delta)$ is the smallest among all $WI(i)$'s where $j \in \{j : j \in I, \max a(IFB(a_i)) < b_j < a_i\}$ when the left endpoint of interval i is being visited. Therefore, invariant 1 is maintained true. This proves the correctness of algorithm MIPD. In practical implementation, $MIPD(i)$'s can be represented by linking lists as follows: we set a pointer from interval i to interval k when we form $MIPD(i)$ by the union of $MIPD(k)$ and $\{i\}$ (line 5). Then $MIPD(i)$ can be obtained by visiting intervals following the pointers from i to 0 and collecting the intervals visited. Thus, the union operation of line 5 can be done in constant time and line 11 can be implemented to run in $O(n)$ time. Since an interval is appended to the tail of L exactly once (line 10), the total number of deletions done by lines 7 and 9 is at most n. Thus we have the following theorem.

THEOREM 2.4. *Given a set I of sorted intervals, a minimum-weight independent dominating set of $G(I)$ is found in $O(n)$ time and $O(n)$ space by Algorithm MIDS.*

2.2. Domination. In this subsection, we describe an algorithm that will be used to solve the weighted domination problem on interval and circular-arc graphs. Let $PD(i)$ be the collection of all PD sets of $G(I)$ whose first and last intervals are intervals 1 and i, respectively. That is, $S \in PD(i)$ if and only if $first(S) = 1$, $last(S) = i$, and S dominates $ISB(b_i)$. For simplicity, we use $MPD(i)$ to denote $\mathrm{Min}(PD(i))$. Then, we have the following lemma.

LEMMA 2.5. *The following two statements are true:*

(1) $MPD(1) = \{1\}$.

(2) *For* $2 \le i \le n$, $MPD(i) = \{i\} \cup \mathrm{Min}\{MPD(j) : \max a(IFB(a_i)) < b_j < b_i\}$.

Proof. Statement (1) is easily seen. We will prove statement (2) by showing that (i) if $S \in PD(k)$ and $\max a(IFB(a_i)) < b_k < b_i$, then $S \cup \{i\} \in PD(i)$, and (ii) there exists a set $S \in PD(i)$ such that $w(S) = w(MPD(i))$, $k = last(S - \{i\})$, $S - \{i\} \in PD(k)$, and $\max a(IFB(a_i)) < b_k < b_i$.

We first prove statement (i). By definition, $last(S) = k$ and S dominates $ISB(b_k)$ since $S \in PD(k)$. By Lemma 2.2, interval i dominates $ISB(b_i) - ISB(b_k)$ since $\max a(IFB(a_i)) < b_k$. Thus $S \cup \{i\}$ dominates $ISB(b_i)$. This proves statement (i).

Next, we prove statement (ii). Since $PD(i) \neq \emptyset$, there exists a set S' such that $S' \in PD(i)$ and $w(S') = w(MPD(i))$. Then we obtain S by removing all intervals contained in interval i except interval 1 from S'; i.e., let $S = S' - \{h : h \in I, h \neq 1, a_i < a_h < b_h < b_i\}$. It is easy to see that $S \in PD(i)$. Clearly $w(S) \le w(S')$ since all weights are nonnegative. Thus $w(S) = w(MPD(i))$. Let interval k be the last interval of $S - \{i\}$. Obviously, $b_k < b_i$ and $a_k < a_i$ if $k \neq 1$. Thus S dominates $ISB(b_i)$, $S - \{i\}$ dominates $ISB(b_k)$, and interval i dominates $ISB(b_i) - ISB(b_k)$ no matter whether interval k is equivalent to interval 1 or not. Since $S - \{i\}$ dominates $ISB(b_k)$, by definition $S - \{i\} \in PD(k)$. By Lemma 2.2, we have that $\max a(IFB(a_i)) < b_k$. This proves statement (ii). □

In the following, we briefly describe the static disjoint set union and find the algorithm proposed by Gabow and Tarjan [15] that is used in solving our problems. The

disjoint set union and find problem [1], [15] is to carry out three kinds of operations on disjoint sets:

$makeset(x)$:	Create a new singleton set $\{x\}$ whose name is x for an element x which is in no existing set.
$find(x)$:	Return the name of the set containing element x.
$union(x, y)$:	Form a new set that is the union of the sets containing x and y, destroying the two old sets. The name of the new set is the name of the old set containing x.

We refer to $union(x, y)$ as the operation that unites the set containing y to the set containing x. The fastest known algorithm for this problem runs in $O(p\alpha(p+q, q)+q)$ time and $O(q)$ space, where α is the inverse of Ackermann's function [37], [38], q is the number of elements, and p is the number of operations performed. A special case of the disjoint set union-find problem can be formalized as follows. We are given a tree T of q nodes that corresponds to the initial q singleton sets. Let the parent of the node v in T be denoted by $parent(v)$. The problem involves performing a sequence of union and find operations such that each union can be only of the form $union(parent(v), v)$. T is called the *static union tree* and the problem is referred to as the *static tree set union*. For this special case, each union and find operation can be supported in $O(1)$ amortized time on a random access machine, and the total space required is $O(q)$ [15]. There is a special case of the static tree set union problem where the union tree is a path [15] also known as *interval union-find problem* [16].

For simplicity, let $W(k) = w(MPD(k))$. Following Lemma 3.2, we design Algorithm MPD for computing $W(i)$ and $MPD(i)$ for all $i \in I$ in $O(n)$ time and space. All $MPD(i)$'s are stored in an $O(n)$ space data structure in the same way as we store $MIPD(i)$'s in the previous subsection. An $MPD(i)$ can be output from this data structure in $O(|MPD(i)|)$ time for all $i \in I$. We refer to a set S of left endpoints of I as a *consecutive left endpoint set* if there exist left endpoints e_1 and e_2 of S such that $S = \{e : e_1 \leq e \leq e_2, e \text{ is a left endpoint of } I\}$. Note that e_1 and e_2 may be identical. Algorithm MPD first scans endpoints to find left endpoint sets $\{a_j : b_{i-1} < a_j < b_i\}$ where $b_0 = 0$ for all $i \in I$. It forms a set for each left endpoint set found by using *makeset* and *union* operations. Note that all these left endpoint sets are either empty or consecutive left endpoint sets. Initially, each left endpoint set $\{a_j : b_{i-1} < a_j < b_i\}$ is associated with right endpoint b_i. What we mean by saying that a set is associated with a right endpoint is that both of the operations of finding the set from the right endpoint and finding the right endpoint from the set can be done in constant time. This can be done easily by pointers. Besides the left endpoint sets, the algorithm also maintains a list L. The elements stored in L are right endpoints. Initially, $L = \{b_1, b_2, \ldots, b_n\}$. Elements of L are stored in a list in increasing order. Details of the algorithm are as follows.

ALGORITHM MPD. Computing $W(i)$ and an $O(n)$ space data structure such that an $MPD(i)$ can be output from this data structure in $O(|MPD(i)|)$ time for all $i \in I$.

Input. A set I of sorted intervals and the weights of intervals.

Output. $W(i)$ and an $O(n)$ space data structure such that an $MPD(i)$ can be output in $O(k)$ time, where $k = |MPD(i)|$, for all $i \in I$.

Method.

1. Find $\max a(IFB(a_i))$ for all $i \in I$;
2. Scan the endpoints of I to find left endpoint sets $\{a_j : b_{i-1} < a_j < b_i\}$ where $b_0 = 0$ for all $i \in I$.

Each right endpoint set b_i is associated with
the left endpoint set $\{a_j : b_{i-1} < a_j < b_i\}$.
3. $L \leftarrow \{b_1, b_2, \ldots, b_n\}$;
4. $W(1) \leftarrow w(1)$; $MPD(1) \leftarrow \{1\}$;
5. **for** $i = 2$ **to** n **do**
6. Find the set containing $\max a(IFB(a_i))$;
7. Let b_k be the right endpoint associated with this set;
8. $MPD(i) \leftarrow \{i\} \cup MPD(k)$;
The union operation is implemented by setting a pointer from i to k.
9. $W(i) \leftarrow w(i) + W(k)$;
10. **while** $W(interval(pred(b_i))) > W(i)$ **do**
11. Unite the set associated with $pred(b_i)$ to the set associated with b_i by union operation;
12. Delete $pred(b_i)$ from L;
13. **end while**
14. **end for**

The correctness of this algorithm can be proved by induction. Algorithm MPD visits intervals one by one (line 5). Let i be the currently visited interval and $L' = \{e : e \in L, e < b_i\}$. Algorithm MPD maintains the following three invariants: immediately before interval i is visited, invariant 1, for any two right endpoints f and h of L', $W(interval(f)) \leq W(interval(h))$ if $f < h$; invariant 2, if the set containing left endpoint e_1 is associated with the right endpoint e_2, then e_2 is the successor of e_1 in L; and invariant 3, for every right endpoint e that is removed from L, there exists a right endpoint e' in L' such that $e < e'$ and $W(interval(e)) > W(interval(e'))$. Initially, all three invariants are true. Then they are maintained by the while loop of lines 10 to 13. By invariant 1 and 2, we have that $MPD(k) = \text{Min}(\{MPD(j) : b_j \in L', \max a(IFB(a_i)) < b_j < b_i\})$ where k is the interval found by line 7. By invariant 3, we have that $MPD(k) = \text{Min}(\{MPD(j) : \max a(IFB(a_i)) < b_j < b_i\})$. By Lemma 2.5, it is easy to see the correctness of Algorithm MPD. Note that the union operation of line 8 is implemented by setting a pointer from i to k. Thus to output $MPD(i)$ we simply follow the pointers to visit intervals starting from interval i until we reach interval 1. The set of intervals visited is an $MPD(i)$. By invariant 2, every left endpoint set formed by line 11 of the algorithm is a consecutive left endpoint set. This is an interval union-find problem [16]. Hence line 11 can be implemented in constant amortized time.

In the following, we give a simple algorithm to find $\max a(IFB(a_i))$ for every interval i in $O(n)$ time.

ALGORITHM P. Computing $\max a(IFB(e))$ for all endpoints of a set I of sorted intervals.

Input. A set I of sorted intervals.

Output. $\max a(IFB(e))$'s of all endpoints of I.

Method.

1. $\beta \leftarrow 0$;
2. **for** $e = 1$ **to** $2n$ **do**
3. $\max a(IFB(e)) \leftarrow \beta$;
4. **if** e is the right endpoint of some interval i, **then**
5. $\beta \leftarrow \max\{a_i, \beta\}$;
end if
end for

We will not prove the correctness of Algorithm P to save space. From the above discussions, it is easy to see that Algorithm MPD can be implemented in $O(n)$ time. Hence we have the following lemma.

LEMMA 2.6. *Given a set I of sorted intervals with weights, Algorithm MPD computes $W(i)$ and $MPD(i)$ for all $i \in I$. All $MPD(i)$'s are stored in an $O(n)$ space data structure such that an $MPD(i)$ can be output in $O(|MPD(i)|)$ time.*

Now we show how to use Algorithm MPD to solve the weighted domination problem on interval graphs. Note the set I_d of intervals is obtained from I as described in the previous subsection. Similarly, a subset S of I is a dominating set of $G(I)$ if and only if $S \cup \{0, n+1\}$ is a dominating set of $G(I_d)$. It is easy to see that interval 0 is the first interval of I_d and all dominating sets of $G(I_d)$ contain intervals 0 and $n+1$. Thus, $MPD(n+1)$ of $G(I_d)$ is a minimum-weight dominating set of $G(I_d)$ and $MPD(n+1) - \{0, n+1\}$ is a minimum-weight dominating set of $G(I)$. Since $MPD(n+1)$ can be computed in $O(n)$ time and space by Algorithm MPD, we have the following theorem.

THEOREM 2.7. *Given a set of sorted intervals, a minimum-weight dominating set of $G(I)$ can be found in $O(n)$ time and $O(n)$ space.*

2.3. Connected domination. If a graph is not connected, then it has no connected dominating set. Whether $G(I)$ is connected can be checked in $O(n)$ time. Hence we assume that $G(I)$ is connected in this subsection. Let $CPD(i)$ be the set of all connected PD sets of $G(I)$ whose first and last intervals are intervals 1 and i, respectively. That is, $CPD(i) = \{S : S \subseteq I, first(S) = 1, last(S) = i,\ G(S)$ is connected, and S dominates $ISB(b_i)\}$. Let $MCPD(i) = \mathrm{Min}(CPD(i))$. If $G(\{1, 2, \ldots, i\})$ is not connected, then $CPD(i) = \emptyset$. In this case, $MCPD(i)$ is considered as a set of infinite weight. The following lemma can be proved by arguments similar to those for proving Lemma 2.5.

LEMMA 2.8. *The following two statements are true:*

(1) $MCPD(1) = \{1\}$; *and*

(2) *For* $2 \le i \le n$, $MCPD(i) = \{i\} \cup \mathrm{Min}(\{MCPD(j) : a_i < b_j < b_i\})$.

Proof. The correctness of statement (1) is easily verified. We will prove statement (2) by showing that (i) if $S \in CPD(k)$ and $a_i < b_k < b_i$, then $S \cup \{i\} \in CPD(i)$; (ii) if $CPD(i) \neq \emptyset$, then there exists a set $S \in CPD(i)$ such that $w(S) = w(MCPD(i))$, $S - \{i\} \in CPD(k)$, where $k = last(S - \{i\})$ and $a_i < b_k < b_i$.

Statement (i) can be easily verified. We prove statement (ii). Let $S' \in CPD(i)$ and $w(S') = w(MCPD(i))$. We obtain S by removing from S' all intervals which are contained in interval i except interval 1. That is, $S = S' - \{j : j \in I, j \neq 1, a_i < a_j < b_j < b_i\}$. Clearly $w(S) \le w(S')$ since all interval weights are nonnegative. It is easy to see that $S \in CPD(i)$ and $w(S) = w(MCPD(i))$. Let $k = last(S - \{i\})$. Obviously, $a_i < b_k < b_i$ since $G(S)$ is connected. By definition, S dominates $ISB(b_i)$. It is easy to verify that $S - \{i\}$ dominates $ISB(b_k)$ and $G(S - \{i\})$ is connected no matter whether interval k is equivalent to interval 1 or not. This proves that $S - \{i\} \in CPD(k)$. □

For simplicity, let $WC(k)$ denote $w(MCPD(k))$. Following Lemma 2.8, one might think that we can modify Algorithm MPD to compute $WC(i)$ and $MCPD(i)$ for all $i \in I$ in $O(n)$ time simply by replacing $\max a(IFB(a_i))$ of line 6 by a_i. We should be careful of that because $2 \le i \le n$, $\{j : j \in I, \max a(IFB(a_i)) < b_j < b_i\}$ is always not empty but $\{j : j \in I, a_i < b_j < b_i\}$ is possibly empty. If $\{j : j \in I, a_i < b_j < b_i\}$ is empty, then $CPD(i) = \emptyset$. Thus $MCPD(i)$ is a set of infinite weight. To modify Algorithm MPD to compute $WC(i)$ and $MCPD(i)$ for all $i \in I$ in $O(n)$ time, we

replace lines 6 to 13 by lines 6' to 17' as follows. Note that lines 8' and 9' handle the case that $\{j : j \in I, a_i < b_j < b_i\} = \emptyset$.

6'. Find the set containing a_i;
7'. Let b_k be the right endpoint associated with the left endpoint set containing a_i;
8'. **if** $k = i$ **then do**
9'. Mark that $MCPD(i)$ is a set of infinite weight;
10'. **else do**
11'. $MCPD(i) \leftarrow \{i\} \cup MCPD(k)$;
The union operation is implemented by setting a pointer from i to k;
12'. $WC(i) \leftarrow w(i) + WC(k)$;
13'. **while** $WC(interval(pred(b_i))) > WC(i)$ **do**
14'. Unite the set associated with $pred(b_i)$ to the set associated with b_i by union operation;
15'. Delete $pred(b_i)$ from L;
16'. **end while**
17'. **end if else**

Thus, we have the following lemma.

LEMMA 2.9. *Given a set I of sorted intervals with weights, we can compute $WC(i)$ and $MCPD(i)$ for all $i \in I$. All $MCPD(i)$'s are stored in an $O(n)$ space data structure such that an $MCPD(i)$ can be output in $O(|MPD(i)|)$ time.*

Let I_c be the set of intervals obtained by augmenting I with two zero-weight intervals, intervals 0 and $n+1$, where $a_0 = 0$, $b_0 = b_1 - 0.5$, $a_{n+1} = \max a(I) + 0.5$, $b_{n+1} = 2n + 1$. Note that $\max a(I)$ denotes the largest left endpoint of intervals in I. It is easy to see that the endpoints of I_c can be sorted in $O(n)$ time and intervals 0 and $n+1$ are the first and last interval of I_c, respectively. It is straightforward to verify that a subset S of I is a connected dominating set of $G(I)$ if and only if $S \cup \{0, n+1\}$ is in $CPD(n+1)$ of $G(I_c)$. Thus, we can find a minimum-weight connected dominating set of $G(I)$ by computing $MCPD(n+1)$ of $G(I_c)$. Since $MCPD(n+1)$ of $G(I_c)$ can be computed in $O(n)$ time, we have the following theorem.

THEOREM 2.10. *Given a set I of sorted intervals, a minimum-weight connected dominating set of $G(I)$ can be found in $O(n)$ time.*

2.4. Total domination. A subset S of I is called an *intermediate total partial dominating* (ITPD) set if the following four conditions are satisfied:

(1) S contains the first interval of I;
(2) S is a PD set of $G(I)$;
(3) $G(S - \{last(S)\})$ has no isolated vertices; and
(4) $\max b(S - \{last(S)\})$ is less than the left endpoint of $last(S)$; i.e., the last interval of S does not overlap any other interval of S.

Let $TPD(i)$ be the collection of all TPD sets which contain the first interval of I and whose last intervals are equivalent to interval i, respectively, let $ITPD(i)$ be the collection of all ITPD sets whose last intervals are equivalent to interval i, and let $XTPD(i) = TPD(i) \cup ITPD(i)$. For simplicity, let $MTPD(i) = \text{Min}(TPD(i))$, $MITPD(i) = \text{Min}(ITPD(i))$, $MXTPD(i) = \text{Min}(XTPD(i))$, $WIT(i) = w(MITPD(i))$, $WT(i) = w(MTPD(i))$, and $WX(i) = w(MXTPD(i))$. Let $X(i) = \cup_{a_i < b_j < b_i} XTPD(j)$, $MX(i) = \text{Min}(X(i))$, $K(i) = \{\{j\} : j \in I, a_i < a_j < b_j < b_i\})$, and $MK(i) = \text{Min}(K(i))$. Note that if $K(i) = \emptyset$, then $MK(i)$ is considered as a set of infinite weight. In words, $X(i)$ is the union of all $XTPD(j)$'s where $a_i < b_j < b_i$ and $K(i)$ is the collection of all single interval sets $\{j\}$'s where $a_i < a_j < b_j < b_i$.

Then, we have the following lemma.

LEMMA 2.11. *The following four statements are true:*

(1) $MTPD(1)$ *is a set of infinite weight, and* $MITPD(1) = \{1\}$;

(2) *for* $2 \le i \le n$ *and* $a_i < b_1$, $MITPD(i)$ *is a set of infinite weight*;

(3) *for* $2 \le i \le n$ *and* $a_i > b_1$, $MITPD(i) = \{i\} \cup \text{Min}(\{MTPD(j) : \max a(IFB(a_i)) < b_j < a_i\})$; *and*

(4) *for* $2 \le i \le n$, $MTPD(i) = \text{Min}(\{MX(i) \cup \{i\}, MITPD(i) \cup MK(i)\})$.

Proof. The correctness of statements (1) and (2) is easy to see. The correctness of statement (3) can be proved by arguments similar to those for proving statement (2) of Lemma 2.3. We prove statement (4) by showing that (i) if $S \in X(i)$, then $S \cup \{i\} \in TPD(i)$; (ii) if $S \in ITPD(i)$ and $K(i) \neq \emptyset$, then $S \cup \{k\} \in TPD(i)$ for any $k \in K(i)$; and (iii) if $TPD(i) \neq \emptyset$, there exists a set $S \in TPD(i)$ which has minimum weight and either $S - \{i\} \in X(i)$ or there exists an interval $k \in S$ such that $\{k\} \in K(i)$ and $S - \{k\} \in ITPD(i)$.

We first prove statement (i). For simplicity, let $k = last(S)$. By definition, $S \in TPD(k)$ or $S \in ITPD(k)$. Clearly, S dominates $ISB(b_k)$. Since $a_i < b_k < b_i$, interval i dominates $ISB(b_i) - ISB(b_k)$. In other words, $S \cup \{i\}$ dominates $ISB(b_i)$. It is straightforward to verify that $G(S \cup \{i\})$ has no isolated vertices. Thus $S \cup \{i\} \in TPD(i)$. This proves statement (i).

Next we prove statement (ii). Since i is the only isolated vertex in $G(S)$, $G(S \cup \{k\})$ has no isolated vertices. By definition, S dominates $ISB(b_i)$. Thus $S \cup \{k\}$ dominates $ISB(b_i)$ too. By definition, $S \cup \{k\} \in TPD(i)$. This proves statement (ii).

Now we prove statement (iii). Let S be a TPD set of $TPD(i)$ such that $w(S) = WT(i)$ and $|S|$ is minimum. By definition, S dominates $ISB(b_i)$. Let $C(i)$ be the set of intervals that are contained in interval i but are not equivalent to interval 1. Consider the following two cases.

Case 1. $C(i) \neq \emptyset$.

Case 2. $C(i) = \emptyset$.

In Case 1, it is easy to see that $S - C(i)$ is still a dominating set of $ISB(b_i)$. Suppose $G(S - C(i))$ has no isolated vertices. Then $S - C(i) \in TPD(i)$. Since interval weights are nonnegative, $w(S - C(i)) \le w(S)$. Apparently, $|S - C(i)| < |S|$. It contradicts the assumption that $S \in TPD(i)$, $w(S) = WT(i)$, and $|S|$ is minimum. Thus, $G(S - C(i))$ has isolated vertices. It is easy to verify that interval i is the only isolated vertex in $G(S - C(i))$ and $1 \notin N(i)$. By definition, $S - C(i) \in ITPD(i)$. Similarly, we can prove that $|S \cap C(i)| = 1$. Since $1 \notin N(i)$, there exists an interval $k \in S$ such that $\{k\} \in K(i)$ and $S - \{k\} \in ITPD(i)$ in this case.

In Case 2, again we consider the following two cases.

Case 2.1. Interval 1 is adjacent to interval i.

Case 2.2. Interval 1 is not adjacent to interval i.

In Case 2.1, it is easy to verify that $S = \{1, i\}$. Thus, $S - \{i\} \in ITPD(1)$. Since $a_i < b_1 < b_i$, we have that $S - \{i\} \in X(i)$.

Now we consider Case 2.2. Let $k = last(S - \{i\})$. Then $a_k < a_i < b_k < b_k$ since $G(S)$ has no isolated vertices and S does not contain any other interval contained in interval i. Thus, $S - \{i\}$ dominates $ISB(b_k)$. Suppose $G(S - \{i\})$ has no isolated vertices. Then, $S - \{i\} \in TPD(k)$. On the other hand, suppose $G(S - \{i\})$ has isolated vertices. It is straightforward to verify that k is the only isolated vertex in $G(S - \{i\})$ and hence $S - \{i\} \in ITPD(k)$. By definition, $S - \{i\} \in X(i)$ since $a_i < b_k < b_i$. This proves statement (iii). □

Based on Lemma 2.11, we design Algorithm MTPD to compute $WT(i)$, $WIT(i)$, $MTPD(i)$, and $MITPD(i)$ for all $i \in I$. All $MTPD(i)$'s and $MITPD(i)$'s are stored

in an $O(n)$ space data structure by using pointers. We can obtain an $MTPD(i)$ (respectively, $MITPD(i)$) from the data structure in $O(|MTPD(i)|)$ (respectively, $O(|MITPD(i)|)$) time. In fact, Algorithm MTPD uses both techniques used by Algorithm MIDS and MPD. Algorithm MTPD visits endpoints one by one and maintains two lists L_1 and L_2. Algorithm MTPD maintains L_1 and L_2 in the same way as Algorithm MIDS and MPD, respectively, maintain list L. Elements in lists L_1 and L_2 are intervals and right endpoints of intervals, respectively. If $j \in L_1$ and a_i is the left endpoint currently visited, then $\max a(IFB(a_i)) < b_j < a_i$. If $e_1 \in L_2$ and $e_1 < e_2$ where e_2 is the endpoint currently visited, then there does not exist a right endpoint $e_3 \in L_2$ such that $e_1 < e_3 < e_2$ and $WX(interval(e_1)) > WX(interval(e_3))$.

ALGORITHM MTPD. Compute $WT(i)$, $WIT(i)$, $MTPD(i)$, and $MITPD(i)$ for all $i \in I$.

Input. A set I of sorted intervals and the weights of all intervals.

Output. $WT(i)$, $WIT(i)$, $MTPD(i)$, and $MITPD(i)$ for all $i \in I$. All $MTPD(i)$'s and $MITPD(i)$'s are stored in an $O(n)$ space data structure. We can obtain an $MTPD(i)$ (respectively, $MITPD(i)$) from the data structure in $O(|MTPD(i)|)$ (respectively, $O(|MITPD(i)|)$) time.

Method.

1. For each $i \in \{j : j \in I - \{1\}, a_j < b_1\}$,
   mark that $MITPD(i)$ is a set of infinite weight;
   $WIT(i) \leftarrow \infty$ for all $i \in \{j : j \in I - \{1\}, a_j < b_1\}$;
2. Mark that $MTPD(1)$ is a set of infinite weight;
   $WT(1) \leftarrow \infty$; $MITPD(1) \leftarrow \{1\}$; $WIT(1) \leftarrow w(1)$;
   $MXTPD(1) \leftarrow \{1\}$; $WX(1) \leftarrow WIT(1)$;
3. $L_1 \leftarrow \{1\}$; $L_2 \leftarrow \{b_1, b_2, \ldots, b_n\}$;
4. For each interval $i \in I - \{1\}$, find $MK(i)$;
5. Scan the endpoints of I to find left endpoint sets
   $\{a_j : b_{i-1} < a_j < b_i\}$ where $b_0 = 0$ for all $i \in I$.
   Each right endpoint b_i is associated with
   the left endpoint set $\{a_j : b_{i-1} < a_j < b_i\}$.
6. **for** $e = b_1 + 1$ **to** $2n$ **do**
7. **if** e is equivalent to the left endpoint of interval i, i.e., $e = a_i$, **then do**
8. $MITPD(i) = \{i\} \cup MTPD(\delta_{L_1})$;
    This statement can be implemented by saving the value of δ_{L_1}.
9. $WIT(i) = w(i) + WT(\delta_{L_1})$;
10. **else** e is the right endpoint of interval i, i.e., $e = b_i$, **do**
11. Find the set containing a_i;
12. Let b_k be the right endpoint associated with the set containing a_i;
13. **if** $k = i$ **then do**
14. Mark that $MTPD(i)$ is a set of infinite weight; $WT(i) \leftarrow \infty$;
15. **else do**
16. $MTPD(i) = \text{Min}(\{MXTPD(k) \cup \{i\}, MITPD(i) \cup MK(i)\})$;
     $WT(i) = \min\{w(i) + WX(j), w(MK(i)) + WIT(i)\}$;
     This statement can be implemented by marking that
     whether $MTPD(i) = MXTPD(k) \cup \{i\}$
     or $MTPD(i) = MITPD(i) \cup MK(i)$.
     If $MTPD(i) = MXTPD(k) \cup \{i\}$, then we also save the value
     of k.
     end if else

17. $MXTPD(i) = \text{Min}(\{MTPD(i), MITPD(i)\})$;
This statement can be implemented by marking that
whether $MXTPD(i) = MTPD(i)$ or $MXTPD(i) = MITPD(i)$.
18. $WX(i) = \min\{WT(i), WIT(i)\}$;
19. Delete all elements of L_1 whose right endpoints are less
than a_i from the head of the list one by one;
20. **while** $L_1 \neq \emptyset$ and $WT(\tau_{L_1}) > WT(i)$ **do**
21. Delete τ_{L_1} from L_1;
end while
22. Append i to the end of L_1;
23. **while** $WX(interval(pred_{L_2}(b_i))) > WX(i)$ **do**
24. Unite the set associated with $pred_{L_2}(b_i)$ to the set associated
with b_i by union operation;
25. Delete $pred_{L_2}(b_i)$ from L_2;
26. **end while**
end if else
end for

It is easy to see the correctness of lines 1 to 3 of Algorithm MTPD. Then it visits endpoints starting from $b_1 + 1$ one by one in increasing order. (See the for loop of lines 6 to 26.) Algorithm MTPD maintains list L_1 in the same way that Algorithm MIDS maintains list L (lines 19 to 22). It maintains list L_2 in the same way that Algorithm MPD maintains list L (lines 23 to 25). For each $MTPD(i)$, we mark whether $MTPD(i) = MXTPD(k) \cup \{i\}$ or $MTPD(i) = MITPD(i) \cup MK(i)$ (line 16). If $MTPD(i) = MXTPD(k) \cup \{i\}$, then we also save the value of k. For each $MITPD(i)$, we save the value of δ_{L_1} (line 8). For each $MXTPD(i)$, we mark whether $MXTPD(i) = MTPD(i)$ or $MXTPD(i) = MITPD(i)$ (line 17). Thus, all $MTPD(i)$'s and $MITPD(i)$'s are stored in an $O(n)$ space. Clearly, we can obtain any $MTPD(i)$ or $MITPD(i)$ recursively according to the information stored. Hence, following Lemma 2.11, it is not hard to see that algorithm MTPD is correct. Algorithm MTPD can be implemented in $O(n)$ time if $MK(i)$'s for all $i \in I$ can be computed in $O(n)$ time. For the cardinality case that $w(i) = 1$ for all $i \in I$, we can compute $MK(i)$'s for all $i \in I$ in $O(n)$ time easily. In the following, we will give an $O(n \log\log n)$ algorithm to compute all $MK(i)$'s for the case that $w(i) = 1$ is not true for all $i \in I$. Algorithm K visits intervals one by one in increasing order of their right endpoints. A set of left endpoints that have been visited are kept in an L. Initially, $L = \emptyset$. Algorithm K maintains the following invariants:

(1) For $e_1, e_2 \in L$, if $e_1 < e_2$, then $w(interval(e_1)) \leq w(interval(e_2))$.

(2) Right before interval i to be visited, if $j < i$ and $a_j \notin L$, then there exists an interval k such that $k \in I$, $k < i$, $a_j < a_k$, $a_k \in L$, and $w(k) < w(j)$.

When the algorithm visits interval i, if $L = \emptyset$ or a_i is greater than the maximum element in L, then interval i does not contain any other interval of I; i.e., $MK(i)$ is a set of infinite weight. Otherwise, $MK(i) = \{k\}$ and $w(MK(i)) = w(k)$ where k is the interval and that a_k is the successor of a_i in L. Details of the algorithm are as follows.

ALGORITHM K. Find $MK(i)$ for every interval i of I.

Input. A set I of sorted intervals and the weights of all intervals.

Output. $MK(i)$'s for all interval $i \in I$.

Method.

```
1.      L = ∅;
2.      for i = 1 to n do
3.          if L = ∅ or a_i > τ then do
4.              Mark that MK(i) is a set of infinite weight;
5.          else do
6.              k ← interval(succ(a_i));
7.              MK(i) ← {k}; w(MK(i)) ← w(k);
            end if else
8.          if L = ∅ or a_i > τ or w(i) ≤ w(interval(succ(a_i))) then do
9.              Insert a_i into L;
10.             while a_i ≠ δ and w(interval(pred(a_i))) > w(i) do
11.                 remove pred(a_i) from L;
                end while
            end if
        end for
```

The two invariants are true initially. Then they are maintained by lines 8 to 11. The data structure L can be implemented by using the two-level priority queue proposed by van Emde Boas [7], which supports the operations of finding the minimum, maximum, predecessor, and successor of an element, inserting and deleting an element in $O(\log\log n)$ time. Hence the running time of Algorithm K is $O(n\log\log n)$. This leads to the following lemma.

LEMMA 2.12. *Given a set I of sorted intervals with weights, Algorithm MTPD computes $WT(i)$, $WIT(i)$, $MTPD(i)$, and $MITPD(i)$ for all $i \in I$ in $O(n\log\log n)$ time. All $MTPD(i)$'s and $MITPD(i)$'s are stored in an $O(n)$ space data structure. We can obtain an $MTPD(i)$ (respectively, $MITPD(i)$) from the data structure in $O(|MTPD(i)|)$ (respectively, $O(|MITPD(i)|)$) time. If $MK(i)$'s for all $i \in I$ can be computed in $O(n)$ time, then the running time of Algorithm MTPD is $O(n)$.*

Let I_t be the set of intervals obtained by augmenting I four zero-weight intervals, -1, 0, $n+1$, and $n+2$, where $a_{-1} < a_0 < b_{-1} < b_0 < 1$, and $2n < a_{n+1} < a_{n+2} < b_{n+1} < b_{n+2}$. We can see that a subset S of I is a total dominating set of $G(I)$ if and only if $S \cup \{-1, 0, n+1, n+2\}$ is a total dominating set of $G(I_t)$. Thus we can find a minimum-weight total dominating set of $G(I)$ by using Algorithm MTPD to compute $MTPD(n+2)$ of $G(I_t)$. Since $MK(i)$'s for all $i \in I$ can be computed in $O(n)$ time if $w(i) = 1$ for all $i \in I$, a minimum-cardinality total dominating set of $G(I)$ can be found in $O(n)$ time. Thus we have the following theorem.

THEOREM 2.13. *Given a set I of sorted intervals, a minimum-weight total dominating set of $G(I)$ can be found in $O(n\log\log n)$ time and $O(n)$ space; and a minimum-cardinality total dominating set of $G(I)$ can be found in $O(n)$ time.*

3. Extensions to circular-arc graphs. In this section, we shall extend the results of the previous sections to solve the weighted domination problems on $G(A)$ given a set A of sorted arcs with real weights. An arc, starting from endpoint h along clockwise direction to endpoint t, is denoted by $[h, t]$. We refer to endpoints h and t as the *head* and *tail* of arc $[h, t]$, respectively. We use "arc" to refer to a member of A and "segment $[c, d]$" to refer to a continuous part of the circle that begins with an endpoint c and ends with d in clockwise direction [34]. Arbitrarily choose an arc from A. Starting from the head of this arc, label endpoints along clockwise direction from 1 to $2n$. Arcs are numbered from 1 to n in increasing order of their tail. Denote the head and tail of arc i by h_i and t_i, respectively. Note that h_i can be larger than t_i, in which case arc $[h_i, t_i]$ extends h_i, $h_i + 1, \ldots, 2n, 1, \ldots, t_i$.

We first study the problem of finding a minimum-weight independent dominating set in a circular-arc graph $G(A)$ given a set A of sorted arcs. One observes that, for any arc x of A, graph $G(A-N(x))$ is an interval graph and arc x is an isolated vertex in $G(A-N(x))$. Moreover, every dominating set of $G(A)$ contains at least one member of $N[x]$. Thus we can find a minimum-weight independent dominating set of $G(A)$ as follows: choosing a vertex x_0 of minimum degree and letting $N(x_0) = \{x_1, x_2, \ldots, x_d\}$ where d is the degree of x_0, we find a minimum-weight independent dominating set of $G(A - N(x_k))$, which is an interval graph, for each arc $x_k \in N[x_0]$. The one with minimum weight is a minimum-weight independent dominating set of $G(A)$. Since $A - N(x_k)$ can be determined in $O(n)$ time, a minimum-weight independent dominating set on $G(A - N(x_k))$ can be computed in $O(n)$ time using the algorithm presented in section 2.1. We have the following theorem.

THEOREM 3.1. *Given a set A of sorted arcs, a minimum-weight independent dominating set of $G(A)$ can be found in $O(m + n)$ time and $O(n)$ space.*

In the following three subsections, we consider the weighted domination, connected domination, and total domination problems on circular-arc graphs. We consider nonnegative weight only. The reasons are explained in section 1. We will give a unified approach to solve these problems in $O(n + m)$ time given a set of sorted arcs with weights. We need the following notation.

For $x \in A$, define $\overline{N}(x)$ to be the set of arcs of A that either contains arc x or is contained in arc x, and define $N_R(x)$ and $N_L(x)$ to be the sets of arcs whose heads and tails are contained in arc x, respectively. Let $A_P(x) = A - \overline{N}(x)$, $A_R(x) = A_P(x) - N_L(x)$, and $A_L(x) = A_P(x) - N_R(x)$. It is straightforward to verify that $A_R(x)$ and $A_L(x)$ are interval graphs. For instance, there are ten arcs shown in Figure 2; i.e., arc 1 = [17, 4], arc 2 = [2, 5], arc 3 = [1, 6], arc 4 = [19, 7], arc 5 = [3, 9], arc 6 = [8, 12], arc 7 = [11, 13], arc 8 = [10, 16], arc 9 = [15, 18], arc 10 = [14, 20]. For arc 3, $N[3] = \{1, 2, 3, 4, 5\}$, $\overline{N}(3) = \{2, 4\}$, $N_R(3) = \{5\}$, $N_L(3) = \{1\}$, $A_P(3) = \{1, 3, 5, 6, 7, 8, 9, 10\}$, $A_L(3) = \{1, 3, 6, 7, 8, 9, 10\}$, and $A_R(3) = \{3, 5, 6, 7, 8, 9, 10\}$.

The following observation plays an important role in our algorithms.

LEMMA 3.2. *Suppose A is an arc model and x_0 is any arc of A. The following three statements are true.*

(1) *There exists a minimum-weight total dominating set S of $G(A)$ such that S contains an arc x in $N[x_0]$ and does not contain any other arc containing arc x.*

(2) *There exists a minimum-weight dominating set S of $G(A)$ such that S contains an arc x of $N[x_0]$ and $S \cap \overline{N}(x) = \emptyset$.*

(3) *There exists a minimum-weight connected dominating set S of $G(A)$ such that S contains an arc x of $N[x_0]$ and $S \cap \overline{N}(x) = \emptyset$.*

Proof. (1) Let S be a minimum-weight total dominating set of $G(A)$ with minimum cardinality. Clearly, $S \cap N[x_0] \neq \emptyset$. There exists an arc $x \in S \cap N[x_0]$ such that x is not contained in any other arc of $S \cap N[x_0]$. Since every arc containing arc x is a neighbor of arc x, x is not contained in any other arc of S.

(2) Let S be a minimum-weight dominating set of $G(A)$ with minimum cardinality. By arguments similar to those for proving statement (1), there exists an arc x such that $x \in S \cap N[x_0]$ and arc x is not contained in any other arc of S. Then every arc of $S \cap \overline{N}(x)$ is contained in arc x. Thus $S - \overline{N}(x)$ is also a dominating set of $G(A)$. Since all weights of arcs are nonnegative, $w(S - \overline{N}(x)) \leq w(S)$. Since $|S - \overline{N}(x)| < |S|$ if $S \cap \overline{N}(x) \neq \emptyset$, we have that $S \cap \overline{N}(x) = \emptyset$. Otherwise, it contradicts the assumption that S is a minimum-weight dominating set with minimum cardinality.

(3) This statement can be proved by arguments similar to those for proving statement (2). ∎

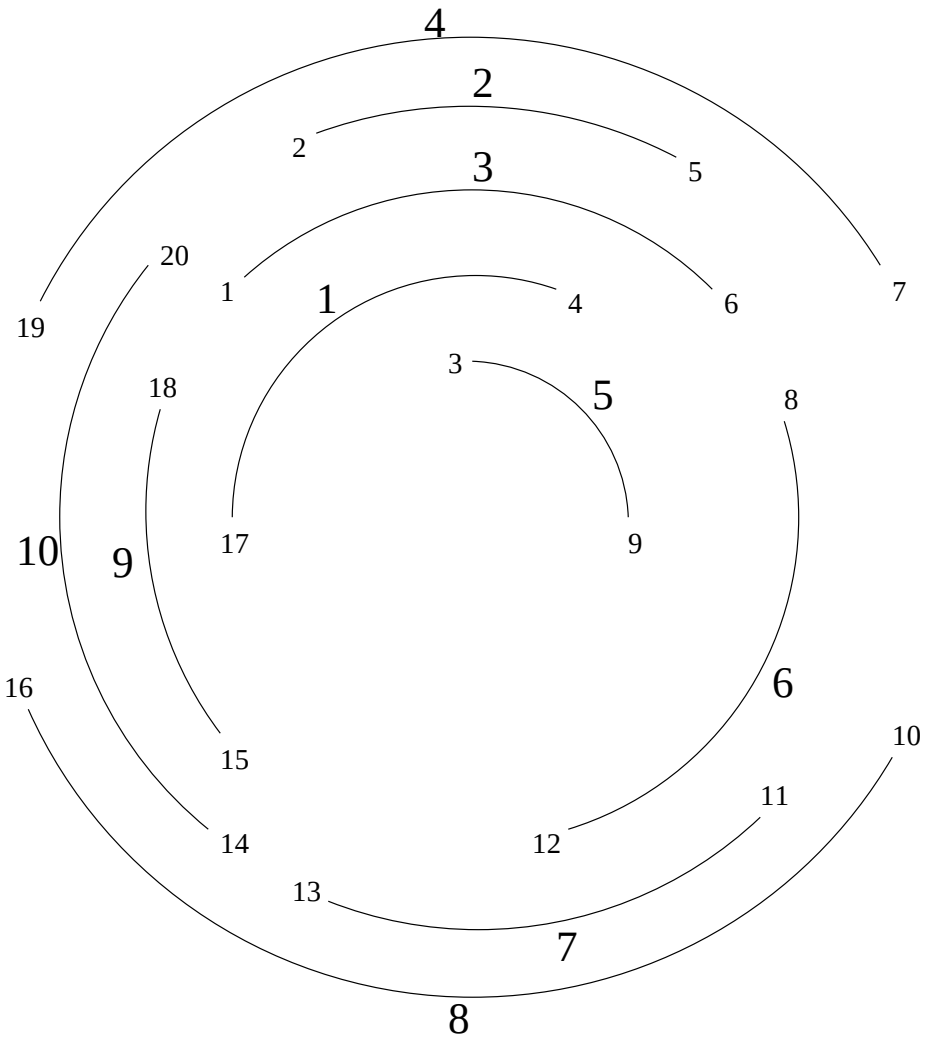


FIG. 2. *A set A of ten arcs.*

Following Lemma 3.2, we define the following:

$D(x) = \{S : S$ is a dominating set of $G(A)$, $x \in S$, and $S \cap \overline{N}(x) = \emptyset\}$,

$CD(x) = \{S : S$ is a connected dominating set of $G(A)$, $x \in S$, and $S \cap \overline{N}(x) = \emptyset\}$, and

$TD(x) = \{S : S$ is a total dominating set of $G(A)$, $x \in S$, x is not contained in any other arc of S$\}$.

For simplicity, let $MD(x) = \text{Min}(D(x))$, $MCD(x) = \text{Min}(CD(x))$, and $MTD(x) = \text{Min}(TD(x))$. By the approach similar to that for independent domination, we can find a minimum-weight dominating set (respectively, connected dominating set, total dominating set) of $G(A)$ given a set A of sorted arcs with nonnegative weights as follows: choosing an arc x_0 of minimum degree and letting $N[x_0] = \{x_0, x_1, x_2, \ldots, x_d\}$ where d is the minimum degree of $G(A)$, we find an $MD(x_k)$ (respectively, $MCD(x)$, $MTD(x)$) for each arc x_k of $N[x_0]$. The one with minimum weight is a minimum-

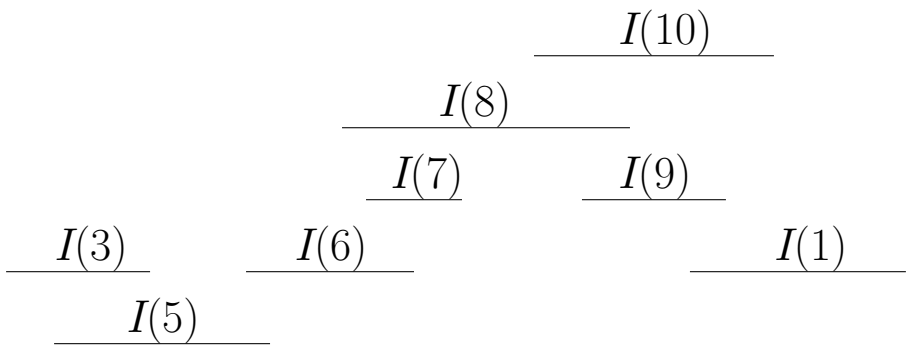


FIG. 3. *The set $I(A_P(3))$ of intervals of the set A of arcs shown in Figure 2.*

weight dominating set (respectively, connected dominating set, total dominating set) of $G(A)$. If $MD(x)$ (respectively, $MCD(x)$, $MTD(x)$) can be found in $O(n)$ time, then a minimum-weight dominating set (respectively, connected dominating set, total dominating set) of $G(A)$ given a set A of sorted arcs with nonnegative weights can be found in $O(n+m)$ time since $O(n(d+1)) = O(n+m)$. In subsections 3.1, 3.2, and 3.3 we show how to find $MD(x)$, $MCD(x)$, and $MTD(x)$ in $O(n)$ time, respectively.

3.1. Domination. In the following, we will give an $O(n)$-time algorithm to find an $MD(x)$. By definition, if $S \in D(x)$, then S is a dominating set of $G(A_P(x))$. In computing $MD(x)$, we first map $A_P(x)$ to a set of intervals. The endpoints of arcs of $A_P(x)$ are numbered in clockwise order from 1 to $2|A_P(x)|$ starting from the head of arc x. Then for every arc $z \in A_R(x)$ we create an interval $I(z) = [h_z, t_z]$; for every arc $z \in N_L(x)$ we create an interval $I(z) = [h_z, t_z + 2|A_P(x)|]$. For Z, a subset of $A_P(x)$, let $I(Z)$ denote $\{I(z) : z \in Z\}$. The above mapping procedure can be done in $O(n)$ time since A is a set of sorted arcs. For example, for the set of ten arcs shown in Figure 2, the set $I(A_P(3))$ of intervals obtained by the above procedure is shown in Figure 3.

The following lemma can be verified easily by the above procedure.

LEMMA 3.3. (1) *$I(x)$ is the first interval of $I(A_P(x))$.*

(2) *For two arcs y and z of $A_P(x)$, arc y overlaps arc z if $I(y)$ overlaps $I(z)$.*

(3) *For $y, z \in A_R(x)$, arc y overlaps arc z if and only if $I(y)$ overlaps $I(z)$.*

(4) *For $y \in A_P(x)$ and $z \in A - N[x]$, arcs y and z overlap if and only if $I(y)$ overlaps $I(z)$.*

We observe that $I(S)$ is a PD set of $I(A_P(x))$ if $S \in D(x)$. Note that PD set was defined in section 2.

LEMMA 3.4. *Suppose $S \subseteq A_P(x)$. Then $S \in D(x)$ if and only if $I(S) \in PD(last(I(S)))$ of $G(I(A_P(x)))$ and $\max b(I(S)) > \max a(I(A - N(x)))$.*

Proof. Suppose $I(S) \in PD(last(I(S)))$ of $G(I(A_P(x)))$ and $\max b(I(S)) > \max a(I(A - N(x)))$. By definition, $I(S)$ dominates all intervals whose left endpoints are less than $\max b(I(S))$. Since $\max b(I(S)) > \max a(I(A - N(x)))$, $I(S)$ dominates $I(A - N(x))$. Because $I(x)$ is the first interval of $I(A_P(x))$, $x \in S$ by definition. By statement (2) of Lemma 3.3, S dominates $A - N(x)$ since $I(S)$ dominates $I(A - N(x))$. Clearly, x dominates $N[x]$ and hence S dominates A. Thus, $S \in D(x)$.

On the other hand, suppose $S \in D(x)$. By definition, $x \in S$ and $S \subseteq A_P(x)$. By statement (4) of Lemma 3.3, $I(S)$ dominates $I(A - N(x))$. Since $I(S)$ does not

dominate any interval whose left endpoint is greater than $\max b(I(S))$, $\max b(I(S)) > \max a(I(A - N(x))$. If $z \in N_R(x)$, then $I(z)$ and $I(x)$ overlap. If $z \in N_L(x)$ and $a_{I(z)} \leq \max b(I(S))$, then $I(z)$ is dominated by $last(I(S))$. Thus all intervals of $I(A_P(x))$ whose left endpoints are less than $\max b(I(S))$ are dominated by $I(S)$. This proves that $I(S) \in PD(last(I(S)))$. □

Lemma 3.4 prompts us to find $MD(x)$ by finding $\text{Min}(\{MPD(i) : b_i > \max a(I(A - N(x)))\})$ from $G(I(A_P(x)))$. By Lemma 2.6, this can be done in $O(n)$ time and space. Thus $MD(x)$ of $G(A)$ can be found an $O(n)$ time and space. This leads to the following theorem.

THEOREM 3.5. *Given a set A of sorted arcs, a minimum-weight dominating set of $G(A)$ can be found in $O(m + n)$ time and $O(n)$ space.*

3.2. Connected domination. In this section, we give an $O(n)$-time algorithm for computing $MCD(x)$. We say a set S of arcs covers the whole circle if every point on the circle is contained in an arc of S. Thus $CD(x)$ can be partitioned into two parts. One part, denoted by $CD_H(x)$, is the collection of those covering the whole circle; the other part, denoted by $CD_B(x)$, consists of those not covering the whole circle. Let $MCD_H(x) = \text{Min}(CD_H(x))$ and $MCD_B(x) = \text{Min}(CD_B(x))$. Clearly, $MCD(x) = \text{Min}(\{MCD_H(x), MCD_B(x)\})$. In the following, we first show how to compute $MCD_H(x)$ in $O(n)$ time and space.

By arguments similar to those for proving Lemma 3.4, we have the following lemma.

LEMMA 3.6. *Suppose $S \subseteq A_P(x)$ and $x \in S$. Then $S \in CD_H(x)$ if and only if $I(S) \in CPD(last(I(S)))$ of $G(I(A_P(x)))$ and $last(I(S)) \in I(N_L(x))$.*

Lemma 3.6 prompts us to find $MCD_H(x)$ by finding $\text{Min}(\{MCPD(i) : i \in I(N_L(x))\})$ from $G(I(A_P(x)))$. By Lemma 2.9, it can be found in $O(n)$ time and space. This leads to the following lemma.

LEMMA 3.7. *Given a set A of sorted arcs, an $MCD_H(x)$ of $G(A)$ can be found in $O(n)$ time and space.*

Next we show how to compute $MCD_B(x)$ in $O(n)$ time and space. For a subset S of A, where $S \neq \emptyset$ and $G(S)$ is connected but S does not cover the whole circle, there exist two arcs u and v of S; arcs u and v may be equivalent such that every arcs of S is contained in segment $[h_u, t_v]$ and every point on segment $[h_u, t_v]$ is contained in at least one arc of S. The head (respectively, tail) of arc u (respectively, v) is called the head (respectively, tail) of S and is denoted by $h(S)$ (respectively, $t(S)$). For example, let $S = \{1, 3, 5, 6\}$ be a subset of arcs shown in Figure 2. Clearly, $G(S)$ is connected and $h(S) = h_1$, $t(S) = t_6$. Suppose $S \in CD_B(x)$, $h(S) = h_u$, and $t(S) = t_v$. Let S_L and S_R denote the sets of arcs contained in segments $[h_u, t_x]$ and $[h_x, t_v]$, respectively. Both $G(S_R)$ and $G(S_L)$ are connected. Clearly, $h(S_R) = h_x$, $t(S_R) = t_v$, $h(S_L) = h_u$, and $t(S_L) = t_x$. For arc $u \in A_L(x)$, define $LCD(u)$ to be the collection of all subsets S's of $A_L(x)$ such that $G(S)$ is connected, and $h(S) = h_u$, $t(S) = t_x$. Similarly, for arc $v \in A_R(x)$, define $RCD(v)$ to be the collection of all subset S's of $A_R(x)$ such that $G(S)$ is connected, and $h(S) = h_x$, $t(S) = t_v$. By definition, if $S \in LCD(u)$ or $S \in RCD(v)$, then $x \in S$. Let $MLCD(u) = \text{Min}(LCD(u))$ and $MRCD(v) = \text{Min}(RCD(v))$. Note that $G(A_R(x))$ is an interval graph and $RCD(v)$ is equivalent to $CPD(v)$ defined in subsection 2.3. By Lemma 2.9, $MRCD(v)$ can be computed in $O(n)$ time for all $v \in A_R(x)$. By symmetric property, $MLCD(u)$ for all $u \in A_L(x)$ can be computed in $O(n)$ time too. Note that we label the endpoints of $A_P(x)$ in clockwise order from 1 to $2|A_P(x)|$ starting from the head of arc x. Suppose $S \in CD_B(x)$, $h(S) = h_u$, and $t(S) = t_v$. Then, $S_L \in LCD(u)$, $S_R \in RCD(v)$, and

no arc of A is contained in segment $[t_v, h_u]$. On the other hand, suppose $u \in A_L(x)$, $v \in A_R(x)$, $h_u > t_v$, and no arc of A is contained in segment $[t_v, h_u]$. If $S_1 \in LCD(u)$ and $S_2 \in RCD(v)$, then $S_1 \cup S_2 \in CD_B(x)$. For arc $u \in A_L(x)$, define $RA(u)$ to be the set of arcs of $A_R(x)$ that are contained in segment $[h_x, h_u]$ if $u \neq x$; and $RA(u) = A_R(x)$ otherwise. And define $\alpha(u) = \max\{h_w : w \in RA(u)\}$. Then for $u \in A_L - \{x\}$, $v \in A_R(x)$, and $t_v < h_u$, there does not exist any arc y such that $t_v < h_y < t_y < h_u$ if and only if $t_v > \alpha(u)$. For arc $v \in A_R(x)$, there does not exist any arc y contained in segment $[t_v, h_x]$ if and only if $t_v > \alpha(x)$. Let $CD_B(x, u) = \{S : S \in CD_B(x), h(S) = h_u\}$. Then we can verify the following four statements easily.

(1) If $u \in A_L(x) - \{x\}$, $S_1 \in LCD(u)$, and $S_2 \in RCD(v)$ where $\alpha(u) < t_v < h_u$, then $S_1 \cup S_2 \in CD_B(x, u)$.

(2) If $v \in A_R(x)$, $t_v > \alpha(x)$, and $S \in RCD(v)$, then $S \in CD_B(x, u)$.

(3) If $S \in CD_B(x, u)$ and $v = t(S)$ where $u \neq x$, then $S_L \in LCD(u)$, $S_R \in RCD(v)$, and $\alpha(u) < t_v < h_u$.

(4) If $S \in CD_B(x, x)$ and $v = t(S)$, then $S_L = \{x\} \in LCD(x)$, $S = S_R \in RCD(v)$, and $\alpha(x) < t_v$.

Let $MCD_B(x, u) = \text{Min}(CD_B(x, u))$. Then $MCD_B(x) = \text{Min}(\{MCD_B(x, u) : u \in A_L(x)\})$. By the above four statements, it is easy to see that $MCD_B(x, u) = MLCD(u) \cup \text{Min}(\{MRCD(w) : w \in A_R(x), \alpha(u) < t_w < h_u\})$ if $u \neq x$; and $MCD_B(x, x) = \text{Min}(\{MRCD(w) : w \in A_R(x), \alpha(x) < t_w\})$ otherwise. Let $MZ(x) = \text{Min}(\{MRCD(w) : w \in A_R(x), \alpha(x) < t_w\})$ and $MZ(u) = \text{Min}(\{MRCD(w) : w \in A_R(x), \alpha(u) < t_w < h_u\})$ for $u \in A_L(x) - \{x\}$. Then $MCD_B(x, u) = MLCD(u) \cup MZ(u)$ and $w(MCD_B(x, u)) = w(MZ(u)) + w(MLCD(u)) - w(x)$ for $u \in A_L(x)$.

In fact, the definition of $\alpha(u)$ is similar to that of $\max a(IFB(e))$ defined in subsection 2.1. Thus we can find an arc v for each arc $u \in A_L(x)$ such that $MZ(u) = MRCD(v)$ in $O(n)$ time by an algorithm similar to Algorithm MIDS. This algorithm scans endpoints of $A_P(x)$ in clockwise order starting from $t_x + 1$ to $2n$. It also maintains a list L of arcs in increasing order of their tails in the same way that Algorithm MIDS maintains a list L of intervals in increasing order of their right endpoints. If arc $v \in L$, then $\alpha(t_u) < t_v < h_u$ and $MRCD(\delta_L) = \text{Min}(\{MRCD(y) : y \in A_R(x), \alpha(t_u) < t_y < h_u\})$ right before the head of arc u is being visited. Details of the algorithm are in the following algorithm.

ALGORITHM MZ. Find an arc v for each arc $u \in A_L(x)$ such that $MZ(u) = MRCD(v)$.

Input. $A_P(x)$ and $w(MRCD(v))$ for all $v \in A_R(x)$.

Method.

1. Number endpoints of $A_P(x)$ from 1 to $2|A_P(x)|$ in clockwise order starting from the head of arc x;
   Number arcs of $A_P(x)$ from 1 to $|A_P(x)|$ in increasing order of their tails;
   $L = \{x\}$;
2. **for** $e = t_x + 1$ **to** $2|A_P(x)|$ **do**
3. **if** e is the head of arc u **then do**
4. $MZ(u) \leftarrow MRCD(\delta)$ where δ is the list head of L;
5. $w(MZ(u)) \leftarrow w(MRCD(\delta))$;
6. **else** e is the tail of arc u **do**
7. Delete all elements of L whose tails are less than h_u from the list head of L one by one;
8. **while** $L \neq \emptyset$ and $w(MRCD(\tau)) > w(MRCD(u))$**do**

9. Delete τ from L where τ is the list tail of L;
end while
10. Append u to the end of L;
end if else
end for
11. $MZ(x) \leftarrow MRCD(\delta)$;

The correctness of Algorithm MZ can be proved by arguments similar to those for proving Algorithm MIDS shown in section 2.1. It is easy to see that the running time of Algorithm MZ is $O(n)$. Thus we have the following lemma.

LEMMA 3.8. *$MCD(x)$ can be computed in $O(n)$ time.*

Since we can compute both $MCD_H(x)$ and $MCD_B(x)$ in $O(n)$ time, we can compute $MCD(x)$ in (n) time. Thus we have the following theorem.

THEOREM 3.9. *Given a set A of sorted arcs, a minimum-weight connected dominating set of $G(A)$ can be found in $O(m+n)$ time and $O(n)$ space.*

3.3. Total domination. In the following, we will focus on how to find $MTD(x)$ in $O(n)$ time given a set A of sorted arcs and an arc $x \in A$. Note that $TD(x)$ is the set of all total dominating sets of $G(A)$ containing arc x but no other arc containing arc x. Define

$TD_1(x) = \{S : S \in TD(x), S \cap (N_L(x) \cup N_R(x)) = \emptyset\}$,
$TD_2(x) = \{S : S \in TD(x), S \cap \overline{N}(x) = \emptyset, S \cap N_L(x) = \emptyset\}$,
$TD_3(x) = \{S : S \in TD(x), S \cap \overline{N}(x) = \emptyset, S \cap N_R(x) = \emptyset\}$,
$TD_4(x) = \{S : S \in TD(x), S \cap \overline{N}(x) = \emptyset, S$ covers the whole circle$\}$, and
$TD_5(x) = \{S : S \in TD(x), S \cap \overline{N}(x) = \emptyset, S \cap N_R(x) \neq \emptyset, S \cap N_L(x) \neq \emptyset, S$ does not cover the whole circle$\}$.

Let $MTD_i(x) = \text{Min}(TD_i(x))$ for $i = 1, 2, 3, 4$, and 5. Suppose $S \in TD(x)$, $S \cap (N_L(x) \cup N_R(x)) \neq \emptyset$, and $S \cap \overline{N}(x) \neq \emptyset$. By definition of $TD(x)$, all arcs of $S \cap \overline{N}(x)$ are contained in arc x. Thus $(S - \overline{N}(x))$ is still a total dominating set of $G(A)$ since $S \cap (N_L(x) \cup N_R(x)) \neq \emptyset$. Hence, $S - \overline{N}(x) \in \cup_{2 \leq i \leq 5} TD_i(x)$. Note that $w(S - \overline{N}(x)) \leq w(S)$ since all weights are nonnegative. In other words, there exists a set $S \in TD(x)$ such that $S \in \cup_{1 \leq i \leq 5} TD_i(x)$ and $w(S) = w(MTD(x))$. Thus we have the following lemma.

LEMMA 3.10. *$MTD(x) = \text{Min}(\{MTD_i(x) : 1 \leq i \leq 5\})$.*

The above lemma suggests that we find $MTD(x)$ by finding MTD_i for $i = 1, \ldots, 5$. It is easy to verify that $TD_4(x) = CD_H(x)$. Thus, $MTD_4(x)$ can be computed in $O(n)$ time by finding $MCD_H(x)$ which was discussed in the previous subsection. In the following, we show how to find $MTD_i(x)$ in $O(n)$ time for $i = 1, 2, 3, 5$. For the reader's convenience, we repeat the definition of $K(x)$ and $MK(x)$ for an interval x of I here: $K(x) = \{\{y\} : y \in I, y \neq x,\ y$ is contained in $x\}$ and $MK(x) = \text{Min}(K(x))$. We will generalize its definition to an arc x of A as follows: $K(x) = \{\{y\} : y \in A, y \neq x,\ y$ is contained in $x\}$ and $MK(x) = \text{Min}(K(x))$. Since $MK(z)$ for all arc $z \in A$ can be found in $O(n+m)$ time, in the following we assume that $MK(z)$ for all arc $z \in A$ are ready for use. To find $MTD_1(x)$, we need the following lemma.

LEMMA 3.11. *The following two statements are true.*

(1) *Suppose S is a total dominating set of $G(A - N[x])$ and y is an arc contained in arc x. Then $\{x, y\} \cup S \in TD_1(x)$.*

(2) *Suppose $S \in TD_1(x)$; then $S - N[x]$ is a total dominating set of $G(A - N[x])$.*

Proof. The lemma is proved by definition. □

By Lemma 3.11, it is easy to see that $\{x\} \cup MK(x) \cup S$ is an $MTD_1(x)$ if S is a minimum-weight total dominating set of $G(A-N[x])$. Since $G(A-N[x])$ is an interval graph, by Theorem 2.13 a minimum-weight total dominating set of $G(A-N[x])$ can be computed in $O(n)$ time since $MK(z)$'s of all arc $z \in A - N[x]$ are ready for use. Hence $MTD_1(x)$ can be found in $O(n)$ time.

In the following, we show how to find $MTD_2(x)$. It is easy to see that $S \subseteq A_R(x)$ if $S \in TD_2(x)$. Clearly, $G(A_R(x))$ is an interval graph. For simplicity, arcs of $A_R(x)$ are considered as intervals in the following lemma where the head and tail of an arc are considered as the left endpoint and right endpoint of its corresponding interval, respectively. Note that $\max a(A_R(x))$ denotes the largest left endpoint of intervals in $A_R(x)$. Also we can see that interval x is the first interval of $A_R(x)$.

LEMMA 3.12. *Suppose $S \subseteq A$. Then $S \in TD_2(x)$ if and only if $S \in TPD(last(S))$ of $G(A_R(x))$ and $b_{last(S)} > \max a(A_R(x))$.*

Proof. Suppose $S \in TD_2(x)$. By definition, $S \subseteq A_R(x)$. Obviously, S is a total dominating set of $G(A_R(x))$ and $x \in S$. Thus, $S \in TPD(last(S))$ and $b_{last(S)} > \max a(A_R(x))$. On the other hand, suppose $S \in TPD(last(S))$ of $G(A_R(x))$ and $b_{last(S)} > \max a(A_R(x))$. Clearly, S is a total dominating set of $G(A_R(x))$, $S \subseteq A_R(x)$, and $x \in S$. Since arc x dominates $N[x]$, S is a total dominating set of $G(A)$. Hence $S \in TD_2(x)$. □

The above lemma suggests that we find $MTD_2(x)$ by finding $\mathrm{Min}(\{MTPD(i) : i \in A_R(x), b_i > \max a(A_R(x))\}$ from $G(A_R(x))$. By Lemma 2.12, it can be done in $O(n)$ time if $MK(z)$'s are known in advance for all $z \in A_R(x)$. Thus $MTD_2(x)$ can be found in $O(n)$ time. By symmetric property, $MTD_3(x)$ can be found in $O(n)$ time in the same way.

In the following, we show how to find $MTD_5(x)$ by using the same technique for computing $MCD_B(x)$. If $S \in TD_5(x)$, then there exists an arc u of S such that h_u is not contained in any other arc of S. Apparently, $u \neq x$. Define $TD_5(x,u) = \{S : S \in TD_5(x), u \in S$, h_u is not contained in any other arc of $S\}$. Let $MTD_5(x,u) = \mathrm{Min}(TD_5(x,u))$. Then, $MTD_5(x) = \mathrm{Min}(\{MTD_5(x,u) : u \in A_L - \{x\}\})$. The definition of $TD_5(x,u)$ is similar to that of $CD_B(x,u)$ defined in the previous subsection. For arc $u \in A_L(x) - \{x\}$, define $LTD(u)$ to be the collection of all subsets S's of $A_L(x)$ such that $x, u \in S$, $G(S)$ has no isolated vertices, all arcs of S are contained in segment $[h_u, t_x]$, and S dominates all arcs that overlap segment $[h_u, t_x]$. Similarly, for arc $v \in A_R(x) - \{x\}$, define $RTD(v)$ to be the collection of all subsets S's of $A_R(x)$ such that $x, v \in S$, $G(S)$ has no isolated vertices, all arcs of S are contained in segment $[h_x, t_v]$, and S dominates all arcs that overlap segment $[h_x, t_v]$. Let $MLTD(u) = \mathrm{Min}(LTD(u))$ and $MRTD(v) = \mathrm{Min}(RTD(v))$. Note that $G(A_R(x))$ is an interval graph and $RTD(v)$ is equivalent $TPD(v)$. By Lemma 2.12, $MRTD(v)$ for all $v \in A_R(x) - \{x\}$ can be computed in $O(n)$ time and space. By symmetric property, $MLTD(u)$ for all $u \in A_L(x) - \{x\}$ can be computed in $O(n)$ time and space too. Note that we number the endpoints of arcs of $A_P(x)$ from 1 to $2|A_P(x)|$ starting from the head of arc x and number the arcs of $A_P(x)$ from 1 to $|A_P(x)|$ starting from arc x in increasing order of their tails. Suppose $S \in TD_5(x)$. Since S does not cover the whole circle, there exist two arcs u and v of S such that $u \in A_L(x) - \{x\}$, $v \in A_R(x) - \{x\}$, $h_u > t_x$, and all arcs of S are contained in segment $[h_u, t_v]$. Let $S_L(u)$ and $S_R(v)$ denote the set of arcs of S contained in segments $[h_u, t_x]$ and $[h_x, t_v]$, respectively. Then we observe that $S_L(u) \in LTD(u)$ and $S_R(v) \in RTD(v)$, and $\alpha(u) < t_v < h_u$. Note that $\alpha(u)$ was defined in the previous subsection. If $u \in A_L(x) - \{x\}$, $S_1 \in LTD(u)$, $v \in A_R(x) - \{x\}$, and $S_2 \in RTD(v)$ where $\alpha(u) < t_v < h(u)$, then $S_1 \cup S_2 \in TD_5(x)$

TABLE 1

	Interval graphs		*Circular-arc graphs*	
	Previous results	New results	Previous results	New results
WIDP	$O(n+m)$	$O(n)$		$O(n+m)$
WDP	$O(n \log n)$ or $O(n+m)$	$O(n)$	$O(n^2 \log n)$	$O(n+m)$
WCDP	$O(n+m)$	$O(n)$		$O(n+m)$
WTDP	$O(n \log n)$ or $O(n+m)$	$O(n \log \log n)$	$O(n^3)$	$O(n+m)$

since $S_1 \cup S_2$ dominates all arcs overlapping segment $[h_u, t_v]$ and there does not exist any arc y such that $t_v < h_y < t_y < h(u)$.

By the above discussions, we have the following lemma.

LEMMA 3.13. *Suppose S is a subset of A and $x \in S$. $S \in TD_5(x,u)$ if and only if there exists an arc v of S such that $S_L(u) \in LCD(u)$, $S_R(v) \in RCD(v)$, and $\alpha(u) < t_v < h(u)$.*

Following the above lemma, we have that $MTD_5(x,u) = MLTD(u) \cup \text{Min}(\{MRTD(v) : v \in A_R(x) - \{x\}, \alpha(u) < t_v < h(u)\})$. Let $MT(u) = \text{Min}(\{MRTD(v) : v \in A_R(x) - \{x\},\ \alpha(u) < t_v < h_u\})$. Thus, $MTD_5(x,u) = MLTD(u) \cup MT(u)$ and $w(MTD_5(x,u)) = w(MLTD(u)) + w(MT(u)) - w(x)$.

We can find an arc $v \in A_R(x)$ such that $MT(u) = MRTD(v)$ for all $u \in A_L(x) - \{x\}$ in $O(n)$ time by an algorithm similar to Algorithm MZ. We omit this algorithm to save space. Then, $w(MTD_5(x,u))$ for all $u \in A_L(x)$ can be computed in $O(n)$ time. Thus, $MTD_5(x)$ can be computed in $O(n)$ time.

We have shown that $MTD_i(x)$ can be computed in $O(n)$ time for $1 \le i \le 5$ excluding the time for computing $MK(z)$ for all $z \in A$. Note that $MK(z)$'s for all $z \in A$ can be computed in $O(n+m)$ time. Thus, a minimum-weight total dominating set of $G(A)$ can be computed in $O(dn) + O(n+m) = O(n+m)$ time. Hence we have the following theorem.

THEOREM 3.14. *Given a set A of sorted arcs, the minimum-weight total dominating set of $G(A)$ can be found in $O(m+n)$ time and $O(n)$ space.*

4. Concluding remarks. In this paper we have presented efficient algorithms to solve the minimum-weight domination problem (WDP) and its three variations, i.e., the minimum-weight independent, connected, and total domination problems (abbreviated by WIDP, WCDP, and WTDP, respectively), on interval and circular-arc graphs. The results of this paper are summarized in Table 1.

It remains an open question whether we can compute a minimum weight total dominating set for a weighted interval graph $G(I)$ of a set of sorted intervals I in $O(n)$ time.

Acknowledgments. The author is grateful to the referees whose extensive comments have led to tremendous improvements in the presentation of the paper.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison–Wesley, Reading, MA, 1974.

[2] T. ASANO, *Dynamic programming on intervals*, Lecture Notes in Computer Science 557, Springer-Verlag, New York, Berlin, 1991, pp. 199–207.

[3] A. A. BERTOSSI, *Total domination in interval graphs*, Inform. Process. Lett., 23 (1986), pp. 131–134.

[4] A. A. BERTOSSI AND M. A. BONUCCELLI, *Some parallel algorithms on interval graphs*, Discrete Appl. Math., 16 (1987), pp. 101–111.

[5] A. A. Bertossi and A. Gori, *Total domination and irredundance in weighted interval graphs*, SIAM J. Discrete Math., 1 (1988), pp. 317–327.
[6] A. A. Bertossi and S. Moretti, *Parallel algorithms on circular-arc graphs*, Inform. Process. Lett., 33 (1989/1990), pp. 275–281.
[7] P. van Emde Boas, *Preserving order in a forest in less than logarithmic time and linear space*, Inform. Process. Lett., 6 (1977), pp. 80–82.
[8] B. Bollobs and E. J. Cockayne, *Graph-theoretic parameters concerning domination, independence, and irredundance*, J. Graph Theory, 3 (1979), pp. 241–249.
[9] M. A. Bonuccelli, *Dominating sets and domatic number of circular-arc graphs*, Discrete Appl. Math., 12 (1985), pp. 203–213.
[10] K. S. Booth and G. S. Lueker, *Testing for consecutive ones property, interval graphs and graph planarity using PQ-tree algorithms*, J. Comput. System Sci., 13 (1976), pp. 335–379.
[11] M. S. Chang, P. Nagavamsi, and C. Pandu Rangan, *Weighted irredundancy in circular-arc graphs*, manuscript, 1995.
[12] E. J. Cockayne and S. T. Hedetniemi, *Towards a theory of domination in graphs*, Networks, 7 (1977), pp. 247–261.
[13] E. M. Eschen and J. P. Spinrad, *An $O(n^2)$ algorithm for circular-arc graph recognition*, in Proc. 4th Annual ACM-SIAM Symposium on Discrete Algorithm, Austin, TX, 1993, pp. 128–137.
[14] M. Farber, *Domination, independent domination and duality in strongly chordal graphs*, Discrete Appl. Math., 7 (1984), pp. 115–130.
[15] H. N. Gabow and R. E. Tarjan, *A linear-time algorithm for a special case of disjoint set union*, J. Comput. System Sci., 30 (1985), pp. 209–221.
[16] Z. Galil and G. F. Italiano, *Data structures and algorithms for disjoint set union problems*, ACM Comput. Surveys, 23 (1991), pp. 319–344.
[17] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, CA, 1979.
[18] F. Gavril, *Algorithms on circular-arc graphs*, Networks, 4 (1974), pp. 357–369.
[19] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
[20] M. C. Golumbic, *Interval graphs and related topics*, Discrete Math., 55 (1985), pp. 113–243.
[21] M. C. Golumbic and P. L. Hammer, *Stability in circular-arc graphs*, J. Algorithms, 9 (1988), pp. 314–320.
[22] U. I. Gupta, D. T. Lee, and J. Y. T. Leung, *Efficient algorithms for interval graphs and circular-arc graphs*, Networks, 12 (1982), pp. 459–467.
[23] W. L. Hsu, *$O(M \cdot N)$ algorithms for the recognition and isomorphism problems on circular-arc graphs*, SIAM J. Comput., 24 (1995), pp. 411–439.
[24] W. L. Hsu and T. H. Ma, *Substitution decomposition on chordal graphs and applications*, Lecture Notes in Computer Science 557, Springer-Verlag, Berlin, New York, 1991, pp. 52–60.
[25] W. L. Hsu and K. H. Tsai, *Linear time algorithms on circular-arc graphs*, Inform. Process. Lett., 40 (1991), pp. 123–129.
[26] N. Korte and R. H. Möhring, *An incremental linear-time algorithm for recognizing interval graphs*, SIAM J. Comput., 18 (1989), pp. 68–81.
[27] R. Laskar, J. Pfaff, S. M. Hedetniemi, and S. T. Hedetniemi, *On the algorithmic complexity of total domination*, SIAM J. Alg. Discrete Math., 5 (1984), pp. 420–425.
[28] G. K. Manacher and T. A. Mankus, *Incorporating negative weight vertices in certain vertex-search graph algorithms*, Inform. Process. Lett., 42 (1992), pp. 293–294.
[29] S. Masuda and K. Nakajima, *An optimal algorithm for finding a maximum independent set of a circular-arc graph* SIAM J. Comput., 17 (1988), pp. 41–52.
[30] G. Ramalingan and C. Pandu Rangan, *Total domination in interval graphs revisited*, Inform. Process. Lett., 27 (1988), pp. 17–21.
[31] G. Ramalingan and C. Pandu Rangan, *A unified approach to domination problems on interval graphs*, Inform. Process. Lett., 27 (1988), pp. 271–274.
[32] G. Ramalingan and C. Pandu Rangan, *New sequential and parallel algorithms for interval graph recognition*, Inform. Process. Lett., 34 (1990), pp. 215–219.
[33] A. Srinivasa Rao and C. Pandu Rangan, *Optimal parallel algorithms on circular-arc graphs*, Inform. Process. Lett., 33 (1989/1990), pp. 147–156.
[34] W. K. Shih, T. C. Chern, and W. L. Hsu, *An $O(n^2 logn)$ time algorithm for the Hamiltonian cycle problem*, SIAM J. Comput., 21 (1992), pp. 1026–1046.
[35] K. Simon, *A new simple linear algorithm to recognize interval graphs*, Lecture Notes in Computer Science 553, Springer-Verlag, Berlin, New York, 1991, pp. 289–308.

[36] A. . Sprague and K. H. Kulkarni, *Optimal parallel algorithms for finding cut vertices and bridges of interval graphs*, Inform. Process. Lett., 42 (1992), pp. 229–234.
[37] R. E. Tarjan, *Efficiency of a good but not linear set union algorithm*, J. Assoc. Comput. Mach., 26 (1975), pp. 215–225.
[38] R. E. Tarjan, J. van Leeuwen, *Worst-case analysis of set union algorithms*, J. Assoc. Comput. Mach., 31 (1984), pp. 245–281.
[39] A. Tucker, *An efficient test for circular-arc graphs*, SIAM J. Comput., 9 (1980), pp. 1–24.
[40] M. S. Yu and C. H. Yang, *A simple optimal parallel algorithm for the minimum coloring problem on interval graphs*, Inform. Process. Lett., 48 (1993), pp. 47–51.

SIAM J. COMPUT.
Vol. 27, No. 6, pp. 1695–1724, December 1998
© 1998 Society for Industrial and Applied Mathematics
009

COMPUTING THE LOCAL CONSENSUS OF TREES*

SAMPATH KANNAN†, TANDY WARNOW†, AND SHIBU YOOSEPH†

Abstract. The inference of consensus from a set of evolutionary trees is a fundamental problem in a number of fields such as biology and historical linguistics, and many models for inferring this consensus have been proposed. In this paper we present a model for deriving what we call a *local consensus tree* T from a set of trees $\mathcal{T}$. The model we propose presumes a function f, called a *total local consensus function*, which determines for every triple A of species, the form that the local consensus tree should take on A. We show that all local consensus trees, when they exist, can be constructed in polynomial time and that many fundamental problems can be solved in linear time. We also consider *partial local consensus functions* and study optimization problems under this model. We present linear time algorithms for several variations. Finally we point out that the local consensus approach ties together many previous approaches to constructing consensus trees.

Key words. algorithms, graphs, evolutionary trees

AMS subject classifications. 05C05, 68Q25, 92-08, 92B05

PII. S0097539795287642

1. Introduction.

An *evolutionary tree* (also called a *phylogeny* or *phylogenetic tree*) for a species set S is a rooted tree with $|S| = n$ leaves labeled by distinct elements in S. Because evolutionary history is difficult to determine (it is both computationally difficult as most optimization problems in this area are *NP-hard* and scientifically difficult as well since a range of approaches appropriate to different types of data exist), a common approach to solving this problem is to apply many different algorithms to a given data set, or to different data sets representing the same species set, and then look for common elements from the set of trees which are returned.

There is extensive literature about inferring consensus from ordered sets of trees, with much attention paid to the properties of the rules for inferring the consensus. In this paper, we will make an explicit assumption that the consensus rule be *independent* of the ordering of the trees in the input; i.e., we will presume that the input to the consensus problem is an unordered multiset of evolutionary trees, each leaf-labelled by the elements in S. We call this input a *profile*, noting that in this paper the terminology is restricted in meaning as we have indicated.

Several consensus methods are described in the literature for deriving one tree from a profile of evolutionary trees. These methods include maximum agreement subtrees [16, 19, 13, 24, 14], strict consensus trees [4, 9], median trees (also known as majority trees) [5], compatibility trees [10, 11, 12], the Nelson tree [22], and the Adams consensus [1].

The algorithms for some of these are implemented in standard packages and are in use; most common, perhaps, are strict and majority consensus tree approaches.

*Received by the editors June 8, 1995; accepted for publication (in revised form) September 12, 1996; published electronically June 3, 1998. The research of the first author was supported in part by NSF grant CCR-9108969. The research of the second author was supported in part by ARO grant DAAL03-89-0031PRI, NSF Young Investigator Award, and by generous support from Paul Angello. The research of the third author was supported in part by ARO grant DAAL03-89-0031PRI, a fellowship from the Institute for Research in Cognitive Science at the University of Pennsylvania, and a fellowship from the Program in Mathematics and Molecular Biology at the University of California at Berkeley, which is supported by NSF grant DMS-9406348.

http://www.siam.org/journals/sicomp/27-6/28764.html

†Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104 (kannan@central.cis.upenn.edu, tandy@central.cis.upenn.edu, yooseph@saul.cis.upenn.edu).

One notion of the *information* content of an evolutionary tree is the degree of resolution indicated by the tree; this can be quantified in a number of ways, for example, by counting the number of internal nodes or the number of resolved triples[1] in the tree. This is because the most usual interpretation of an unresolved triple in an evolutionary tree is that the evolutionary history of that triple cannot be absolutely inferred from the data. Thus, for example, a completely resolved tree (i.e., a binary tree) asserts a hypothesis about the evolution of all triples of taxa, while the star (i.e., root with all taxa children of the root) does not assert any hypothesis about the evolution of any triple. One of the motivations for proposing this new model of consensus tree construction is the observation that on some data sets the strict and majority consensus trees may be fairly uninformative (i.e., be fairly unresolved).

In this paper, we propose a new model, called the *local consensus*. This model is based upon functions, called *local consensus functions*, for inferring the rooted topology of the homeomorphic subtree induced by triples of species. We will show that given any local consensus function, we can determine whether a tree (called the *local consensus tree*) consistent with the constraints implied by the local consensus function can be computed in polynomial time and that many of the natural forms of the local consensus can be computed in linear time. We also analyze optimization problems based upon partial local consensus rules and show that many of these can also be solved in polynomial time. We will show that this method unifies many of the previously favored approaches while providing greater flexibility to the biologists in the interpretation of the data. Furthermore, the local consensus trees produced are, in most cases, significantly more informative (in the sense of more refined; see the above discussion) than trees produced using the strict or majority consensus methods.

2. Preliminaries.

2.1. Trees. Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of species. An *evolutionary tree* for S (also known as a phylogenetic tree or, more simply, a phylogeny) is a rooted tree T with n leaves each labeled by a distinct element from S. The internal nodes denote ancestors of the species in S. For an arbitrary subset $S' \subset S$ we denote by $\mathbf{T}|\mathbf{S'}$ the homeomorphic subtree of T induced by the leaves in S'. In particular, for a specified triple $\{a, b, c\} \subset S$ we denote by $\mathbf{T}|\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ the homeomorphic subtree of T induced by the leaves labeled by a, b, and c. This topology is completely determined by specifying the pair of species among a, b, and c whose least common ancestor (LCA) lies farthest away from the root. If (a, b) is this pair then we denote this by $((a, b), c)$, and T is said to be *resolved* on the triple a, b, c. If T is not binary it may happen that all three pairs of species have the same LCA. In this case we will say that a, b, c is *unresolved* in T and denote this topology by (a, b, c). In this paper, when we say a triple a, b, c is resolved, we mean that $T|\{a, b, c\}$ is one of $((a, b), c), ((a, c), b)$, or $((b, c), a)$.

For a *profile* P, which is defined by a multiset $\{T_1, T_2, \ldots, T_k\}$, we let $P|\{a, b, c\}$ denote the multiset $\{T_1|\{a, b, c\}, T_2|\{a, b, c\}, \ldots, T_k|\{a, b, c\}\}$.

Given a tree T containing nodes u, v, w, we let $lca_T(u, v, w)$ denote the LCA of u, v, and w in T. Also, we let $u \leq_T v$ denote that v is on the path from u to the root of T.

2.2. Local consensus functions, rules, and trees. Let $\mathcal{T}(a, b, c)$ denote the set of rooted subtrees on the leaf set $\{a, b, c\} \subseteq S$; thus $|\mathcal{T}(a, b, c)| = 4$, with three of

[1] See section 2.1 for definitions of a *resolved* triple and an *unresolved* triple.

the trees being resolved and one being the star (i.e., unresolved) tree on a, b, c.

A *local consensus function* is a function f which specifies the constraints for certain (i.e., perhaps not all) triples a, b, c of species. Let A be the set of all three element subsets of S. We define $f : A \to \cup_{\{a,b,c\} \in A} \mathcal{T}(a, b, c) \cup \{*\}$. When $f(X) = *$, for some $X = \{a, b, c\} \in A$, this indicates that the form of the triple a, b, c is *unconstrained*. When $f(X) \neq *\forall X \in A$, i.e., no triple is unconstrained, then f is said to be a *total* local consensus function. Otherwise, f is said to be a *partial* local consensus function.

A rooted tree T (if it exists) which is leaf-labelled by elements from S and which meets all the constraints implied by the local consensus function f is called an *f-local consensus tree*.[2] Note that when a triple a, b, c is set to be unconstrained by f, then $T|\{a, b, c\}$ can be any of the elements in $\mathcal{T}(a, b, c)$. Thus T is a tree such that for all triples $X \in A, T|X = f(X)$, if $f(X) \neq *$.

A local consensus function can be applied to a profile P. It is also possible for the local consensus function to define the form of the output triple based upon the forms the triple takes in the profile. Such local consensus functions are called *local consensus rules*. Let $\mathcal{M}$ be the set of all multisets of size k, where each element of a multiset belongs to $\mathcal{T}(a, b, c)$. A local consensus rule is a function $f : \mathcal{M} \to \mathcal{T}(a, b, c) \cup \{*\}$. If $f(X) = *$, for some $X \in \mathcal{M}$, then f is said to be a partial local consensus rule; otherwise, f is a total local consensus rule.

Given a profile P and a local consensus rule f, the f-local consensus tree (if it exists) is a rooted tree T such that for all triples $X \subseteq S, T|X = f(P|X)$, if $f(P|X) \neq *$.[3]

It is not the case that a local consensus tree necessarily exists for an arbitrary local consensus function (or rule) applied to an arbitrary input profile. Determining whether a local consensus tree exists, and constructing it when it does, is the subject of this paper.

The structure of the paper is as follows. In section 3, we will describe some general techniques for determining if a local consensus tree exists. In particular, we will give a polynomial time algorithm (based upon the algorithm in [3]), which can determine if a local consensus tree exists for an arbitrary local consensus function (or rule), and construct it when it does. We will also describe a class of natural local consensus rules and describe general techniques for constructing local consensus trees from such natural local consensus rules when they exist. In section 4, we then describe some specific natural local consensus rules and some fast algorithms for constructing the local consensus trees. In section 5, we consider optimization problems related to constructing local consensus trees and present efficient algorithms to solve some of these optimization problems. We conclude in section 6 with a discussion and suggestions for extensions.

3. Techniques.

3.1. General local consensus functions. For an arbitrary local consensus function f and an arbitrary profile of trees $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$, we can compute the constraint indicated by f for every triple of species a, b, c. This produces a set of $O(n^3)$ constraints on the consensus tree we wish to construct, where each constraint is a rooted tree for a triple on a species set a, b, c. This rooted tree may be resolved

[2] We will also sometimes refer to it simply as a local consensus tree.

[3] Note that f is defined the same on all triples $X \subseteq S$. As defined above, the triple labels a, b, c serve merely as place holders. The definition of a local consensus rule can easily be changed to accommodate a *different* rule for each triple.

(i.e., it may be of the form $((a,b)c)$) or it may be unresolved (i.e., of the form (a,b,c)). If there is a tree T meeting all these constraints, then T is the local consensus tree for f. Thus, we can reduce the problem of consensus tree construction for an arbitrary local consensus function to the problem of determining consistency of a set of rooted triples.

3.1.1. Rooted triple consistency. We present results related to this general problem.

THEOREM 3.1. *Determining if a tree T exists which meets a set of constraints (and constructing it if it does) can be solved in $O(pn \log n)$ time if the constraints include unresolved triples and otherwise can be solved in $O(pn)$ time, where p is the number of constraints defined by f.*

Proof. In [3], Aho et al. describe algorithms which determine if a family of constraints on LCA relations can be satisfied within a single rooted tree. We describe here the simple algorithm they give for the case where the constraints are given as rooted resolved triples $((x,y),z)$. For such input the algorithm works top-down figuring out the clusters at the children of the root before recursing. To do this the algorithm maintains disjoint sets. Initially all leaves are in singleton sets. For each rooted triple $((x,y),z)$ the algorithm unions the sets containing x and y to indicate that x and y must lie below the same child of the root. This algorithm never unions sets unless this is forced. Recursive calls include constraints that are on species entirely contained in the same component discovered in the previous call. If all the species are seen to be in the same component (either initially or during a recursive call), the algorithm determines that the constraints cannot be simultaneously satisfied. This simple algorithm has a worst-case behavior of $O(pn)$, where there are p LCA constraints and the underlying set S has n elements which will be leaves in the final tree. ∎

However, we can also solve the consistency problem faster than by using the Aho et al. algorithm. In [21], an algorithm is given for the problem addressed in [3] for the case where all the triples are resolved. In this case a faster algorithm can be obtained.

LEMMA 3.1 (Henzinger, King, and Warnow [21]). *Let $\mathcal{A}$ be a set of p resolved rooted triples on a leaf set S with $|S| = n$. We can determine in $\min\{O(p\sqrt{n}), O(p + n^{2.5})\}$ time whether a tree T exists such that $T|\{a,b,c\}$ is homeomorphic to the rooted triple(s) in A on $\{a,b,c\}$ (if such a triple exists in A).*

In the context of the rooted triple consistency problem, we also refer to the work of [8, 7], where the conditions necessary for a given set of triple constraints to define a tree are investigated.

3.2. Constructing local consensus trees in polynomial time. As a consequence of the results in the previous section, we can prove the following theorem.

THEOREM 3.2. *Let f be an arbitrary partial local consensus rule and $\mathcal{T}$ a set of k evolutionary trees on S with $|S| = n$.*

1. *If every triple which is not set to $*$ is defined to be resolved by f, then we can determine if the local consensus tree exists and construct it if it does in $O(kn^3)$ time.*
2. *If f defines some triples (which are not set to $*$) to be unresolved, then we can determine if the local consensus tree exists and construct it if it does in $O(kn^3 + n^4 \log n)$ time.*

Proof. Given f, $\mathcal{T}$, and a triple A, we can determine the form of $\mathcal{T}_f|A$ (for those triples A for which $T_f|A$ is not unconstrained) in $O(kn^3)$ time. If all the triples which are not set to unconstrained are defined to be resolved, then by Lemma 3.1 we can determine if the partial local consensus tree exists and construct it if it does, in

$O(n^{2.5} + p)$ time, where p is the number of constraints. The total time is therefore bounded by the cost of computing the triples. If some of the triples are unresolved then we can use Theorem 3.1 to get an $O(kn^3 + n^4 \log n)$ algorithm which will determine if the tree exists and construct it when it does. □

3.2.1. Constructing local consensus trees from total local consensus rules. While local consensus trees can be constructed in $O(kn^3)$ time from partial local consensus rules, local consensus trees can be computed even faster when the local consensus rule is *total*.

LEMMA 3.2 (Kannan, Lawler, and Warnow [18]). *Given an oracle $\mathcal{O}$ which can answer queries of* "What is the form of $T|\{a, b, c\}$ for a species set $\{a, b, c\}$?"*, we can construct in $O(n^2)$ time a tree T consistent with all the oracle queries (if it exists) and $O(rn \log n)$ time if the tree T has degree bounded by r.*

THEOREM 3.3. *Let f be a total local consensus rule. Then given a set of k rooted trees on n species, we can construct in $O(kn^2)$ time the f-local consensus tree $\mathcal{T}_f$ if it exists. If f always returns resolved subtrees, then we can compute $\mathcal{T}_f$ in $O(kn \log n)$ time.*

Proof. We can implement the oracle determining the form of the homeomorphic subtree of $\mathcal{T}_f$ on a triple a, b, c by first preprocessing the trees to answer LCA queries in constant time using [20]. Then, answering a query needs only $O(k)$ time. By [18], we need only $O(n^2)$ queries and $O(n^2)$ additional work for a total cost of $O(kn^2)$ in the general case. When $\mathcal{T}_f$ has degree bounded by r, we have total cost $O(krn \log n)$. If f always returns resolved subtrees, then $\mathcal{T}_f$ will be binary, so that the total cost is $O(kn \log n)$. □

Note, however, that this algorithm does not verify that the tree constructed is the local consensus tree; that is, it is possible that the constraints are inconsistent, so that no local consensus tree exists for that local consensus function (or rule). When it does, however, the tree constructed will equal the local consensus tree. Thus, when it can be shown that the local consensus tree *does* exist, then this method will necessarily produce the local consensus tree. In general, however, it will be necessary to verify that the constructed tree is the local consensus tree.

We have described two algorithms for inferring whether a local consensus tree exists for an arbitrary local consensus function (or rule). When the local consensus function (or rule) is total, if the local consensus tree exists, it can be constructed in $O(kn^2)$ time, where k is the number of trees in the profile and n is the number of leaves in each tree. However, the tree that results then needs to be verified to be the local consensus tree (and the fastest verification algorithm may still require $\Omega(kn^3)$ time). When the local consensus function (or rule) is partial, then a slower $O(kn^3)$ algorithm can be used, but it simultaneously constructs and verifies that the constructed tree is the local consensus tree.

3.3. Local consensus rules. A local consensus rule must handle essentially three types of situations for each pattern of subtrees in the profile for a triple a, b, c of species: *profile constant on a,b,c; profile compatible on a,b,c; profile incompatible on a,b,c.* The profile of trees may agree on that set a, b, c, and thus all reflect the same evolutionary history, or the trees may differ (in two different ways) on the triple. Depending upon the pattern of different subtrees, the local consensus rule may elect to constrain the form of the output or to leave the output unconstrained for that triple. However, we will only consider a local consensus rule to be natural if it is *conservative*, where by conservative we mean the following definition.

DEFINITION 3.1. *Let P be a profile of evolutionary trees and f be a local consensus rule. Then f is said to be* conservative *for every triple a, b, c, iff, $f(P|\{a, b, c\}) = ((a, b), c)$, then a, b, c is not resolved as $((a, c), b)$ or $((b, c), a)$ in any of the trees in P.*

Being conservative is obviously a natural requirement, since to enforce a topological constraint which is contradicted in the profile is clearly unmotivated.

We now describe the three general scenarios that may arise and discuss the possible constraints that may arise under natural local consensus rules.

Profile constant on a, b, c. If all the trees in the profile have the same form on a triple a, b, c, then we say the profile is *constant* on a, b, c. In this case, a natural local consensus rule should either require that the consensus tree have the same form as the trees in the profile, or it may leave the form unconstrained.

Profile compatible on a, b, c. If all the trees in the profile that have resolved subtrees for a, b, c have the *same* resolved form (i.e., no two trees in the profile resolve a, b, c differently), then the profile is said to be *compatible* on a, b, c. In this case, the natural local consensus rule may elect to leave the tree unconstrained for a, b, c; otherwise, it should constrain the output to either be the unique resolution indicated by the profile or should constrain it to be *unresolved*. In the first case, we call the local consensus rule *optimistic*, and in the second case we call the local consensus rule *pessimistic*.

Profile incompatible on a, b, c. The remaining case is where the profile contains trees which have different resolutions for a, b, c. In this case, a natural local consensus rule may elect to require the consensus tree to be unresolved, or it may select one of the resolutions represented in the profile[4] (perhaps selecting the resolution with the plurality representation), or it may not constrain the output at all.

A local consensus rule can be defined by deciding how it will respond to each of the different situations that can arise. Thus, for example, a natural local consensus rule may require that when the profile is constant on a, b, c, then the output tree is constrained to have that same form, and it may elect to be optimistic in the presence of compatible forms on a, b, c but may leave unconstrained any triple for which the profile is incompatible.

In all of our following discussions, we restrict ourselves to profiles of two trees. The techniques and most observations can be generalized.

4. Specific total local consensus rules. As examples of natural local consensus rules, we will define two total local consensus rules: the *optimistic local consensus (OLC) rule* and the *pessimistic local consensus (PLC) rule*. These are not the only natural local consensus rules that are worthy of study, but the techniques used for constructing local consensus trees for these rules are indicative of general approaches for greatly speeding up the construction and verification phases used in the previous section.

When the trees are not necessarily binary, the local consensus rule may encounter triples for which the profile is not constant but is nevertheless compatible. Because a total local consensus rule must constrain the form of each triple for the consensus tree, it must determine whether to require that the rooted triple be resolved or unresolved. This decision is based upon the interpretation of an unresolved triple, which can be made in one of two ways: *any resolution of the three-way split is possible* or *the unresolved triple indicates a three-way speciation event*. If the local consensus rule

[4]In this case the *conservative* nature of the rule need not be maintained.

chooses to interpret lack of resolution as being consistent with any resolution, then it will constrain the output to be resolved according to the unique resolution present in the profile, and otherwise it will constrain the output to be unresolved. The first type of total local consensus rule is said to be *optimistic* and the second type *pessimistic*.

We now define these two consensus rules.

DEFINITION 4.1. *Let T_1 and T_2 be two rooted trees on the same leaf set S. A rooted tree T is called the* OLC *of T_1 and T_2 iff for each triple a, b, c, $T|\{a, b, c\} = ((a, b), c)$ iff $T_i|\{a, b, c\} = ((a, b), c)$ and $T_j|\{a, b, c\} = ((a, b), c)$ or (a, b, c) for $\{i, j\} = \{1, 2\}$.*

DEFINITION 4.2. *Let T_1 and T_2 be two rooted trees on the same leaf set S. A rooted tree T is called the* PLC *of T_1 and T_2 iff for each triple a, b, c, $T|\{a, b, c\} = ((a, b), c)$ iff $T_1|\{a, b, c\} = T_2|\{a, b, c\} = ((a, b), c)$.*

In the next two subsections we discuss efficient algorithms for these rules. But first we give some basic and standard definitions.

DEFINITION 4.3. *Let T be a rooted tree with leaf set S. Given a node $v \in V(T)$, we denote by $\mathcal{L}(T_v)$ the set of leaves in the subtree T_v of T rooted at v. This is also called the cluster at v and is represented by α_v. The set $C(T) = \{\alpha_v : v \in V(T)\}$ is called the* cluster encoding of T.

Every rooted tree in which the leaves are labeled by S contains all singletons and the entire set S in $C(T)$; these clusters are called the *trivial clusters*. We define a *maximal* cluster to be the cluster defined by the child of the root. (Here we allow for a maximal cluster to be defined by a leaf also.)

We also define the notion of compatibility of a set of clusters.

DEFINITION 4.4. *A set A of clusters is said to be* compatible *iff there exists a tree T such that $C(T) = A$.*

The following proposition can be found in [17].

PROPOSITION 4.1. *A set A of clusters is compatible iff $\forall \alpha_i, \alpha_j \in A$, $\alpha_i \cap \alpha_j \in \{\alpha_i, \alpha_j, \emptyset\}$.*

We now state a theorem which will be used in the later sections.

THEOREM 4.1. *Let T_1 and T_2 be two rooted trees on the same leaf set S and let f be a conservative local consensus rule. If the f-local consensus tree T exists, then $C(T) \cup C(T_1)$ and $C(T) \cup C(T_2)$ are compatible sets.*

Proof. Suppose not and suppose without loss of generality that $C(T) \cup C(T_1)$ is not a compatible set. Then by Proposition 4.1, $\exists \alpha \in C(T)$ and $\beta \in C(T_1)$ such that $\alpha \cap \beta \notin \{\alpha, \beta, \emptyset\}$. Pick $a \in \alpha \cap \beta$, $b \in \alpha - \beta$ and $c \in \beta - \alpha$. The topology of the triple a, b, c in T_1 is $((a, c), b)$ while in T it is $((a, b), c)$. Since f is a conservative local consensus rule, this is impossible. ∎

4.1. OLC. In this section we look at the problem of finding the OLC tree of two trees defined in the previous section. Note that the OLC of two trees may not exist. See Figure 1 for an example.

4.1.1. Characterization of the OLC tree. The following lemma characterizes the OLC tree when it exists.

THEOREM 4.2. *Let T_1 and T_2 be two rooted trees on the same species set S. If the OLC tree T_{olc} exists, then $C(T_{olc}) = A$, where $A = \{\alpha^* \mid \alpha^* = \alpha_1 \cap \alpha_2$, where $\alpha_1 \in C(T_1)$ and $\alpha_2 \in C(T_2)$, and α^* is compatible with both $C(T_1)$ and $C(T_2)\}$.*

Proof. Pick any cluster $\alpha \in A$. If we look at any triple x, y, z with $x, y \in \alpha$ and $z \notin \alpha$, then this triple will be resolved as $((x, y), z)$ in one tree and will be either resolved the same or unresolved in the other tree. In either case, $\alpha \in C(T_{olc})$.

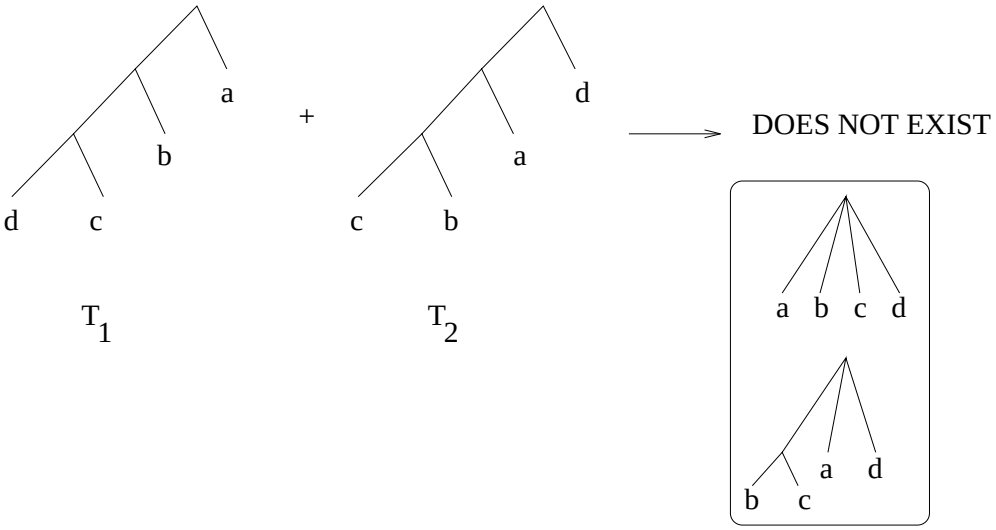


FIG. 1. *Example showing that the OLC need not always exist. The trees in the box are possible candidates, but they each fail to maintain the necessary topology for some triple.*

Conversely, pick any cluster $\alpha \notin A$. There are two cases here, namely, the case when α is not compatible with at least one of $C(T_1)$ and $C(T_2)$ and the case when α is compatible with both $C(T_1)$ and $C(T_2)$.

Now, when α is not compatible with at least one of $C(T_1)$ and $C(T_2)$, using Theorem 4.1, we observe that $\alpha \notin C(T_{olc})$.

For the second case, pick those smallest clusters $\alpha_1 \in C(T_1)$ and $\alpha_2 \in C(T_2)$ such that $\alpha \subseteq \alpha_1$ and $\alpha \subseteq \alpha_2$. (Note that the nodes v and u defining the clusters α_1 and α_2, respectively, are the LCAs in T_1 and T_2, respectively, of the species in α.) Since α_1 and α_2 are the smallest clusters in T_1 and T_2, respectively, containing α and since α is compatible with both $C(T_1)$ and $C(T_2)$, this implies that α is the union of clusters of at least two children of v and also the union of clusters of at least two children of u. Moreover, $\exists a, b \in \alpha$ such that $v = lca_{T_1}(a, b)$ and $u = lca_{T_2}(a, b)$. Furthermore, $\exists \beta \subseteq S, \beta \neq \emptyset$, such that $\alpha_1 \cap \alpha_2 = \alpha \cup \beta$. Thus we can pick a $c \in \beta$ and we have that $T_1|\{a, b, c\} = T_2|\{a, b, c\} = (a, b, c)$. But the topology given by having $\alpha \in C(T_{olc})$ is $((a, b), c)$. Thus $\alpha \notin C(T_{olc})$. □

4.1.2. Construction phase. Since the OLC rule is conservative, if the tree T_{olc} exists, then $C(T_{olc}) \cup C(T_1)$ is a compatible set of clusters, and hence there exists a tree T^* satisfying $C(T^*) = C(T_1) \cup C(T_{olc})$. If we can construct T^* by refining T_1, we can then reduce T^* by contracting all the unnecessary edges and thus obtain T_{olc}. This is the approach we will take.

Note that this approach breaks the construction into two stages: *refinement* and *contraction*.

DEFINITION 4.5. *We say that a tree T_1 is a refinement of tree T_2 if T_2 can be obtained from T_1 by a sequence of edge contractions.*

Refining T_1. The main objective is to refine T_1 so as to include all the clusters from T_{olc}. Before we explain how we do this precisely, we will introduce some notation and lemmas from previous works which enable us to do this efficiently.

DEFINITION 4.6. *Let v be an arbitrary node in a tree T with children $u_1, \ldots, u_k$. A* representative set *of v is any set $\{x_1, x_2, \ldots, x_k\}$ such that $x_i \in \alpha_{u_i}$. We denote by $rep(v)$ one such representative set.*

LEMMA 4.1. *If the OLC tree T_{olc} of trees T_1 and T_2 exists and $v \in T_1$, then $T_{olc}|rep(v)$ is isomorphic to $T_2|rep(v)$.*

Proof. The proof follows from the fact that $T_1|rep(v)$ is a star. □

DEFINITION 4.7. *Let v be a node in a tree T with children $u_1, u_2, \ldots, u_k$. Then $N(v)$ is the subtree induced by $\{v, u_1, u_2, \ldots, u_k\}$.*

We will do the refinement as follows. We will modify the tree T_1^*, where T_1^* is initialized to T_1. In a postorder fashion, for every $v \in V(T_1)$ with representative set $\{x_1, x_2, \ldots, x_k\}$, identify $v^* = lca_{T_1^*}(\alpha_v)$. It can be seen that v^* also has the same number of children as v (since the processing is done in a postorder fashion). Say these are $u_1, u_2, \ldots, u_k$. Replace the subtree $T(v^*)$, rooted at v^* in the following manner: we replace $N(v^*)$ by an isomorphic copy of $T_2|rep(v)$. Next, we replace x_i by the subtree of T_1^* rooted at u_i.

Let T^* be the tree that is produced after considering all the nodes in T_1.

THEOREM 4.3. *Let T_1, T_2 be given and suppose T_{olc} exists. Then the tree T^* that is produced from the algorithm described in the previous paragraph satisfies $C(T^*) = C(T_1) \cup C(T_{olc})$.*

Proof. Since $C(T_{olc}) \cup C(T_1)$ is compatible, all we need to show is that $T_{olc}|rep(v)$ cannot be a proper refinement of $T_2|rep(v)$. If it were, then for some $\{a, b, c\} \subseteq rep(v)$, $T_{olc}|\{a, b, c\}$ would be resolved while $T_2|\{a, b, c\}$ is unresolved. Since $\{a, b, c\} \subseteq rep(v)$, $T_1|\{a, b, c\}$ is also unresolved, forcing T_{olc} to be also unresolved. □

Note that we have reduced the problem of constructing T^* to the problem of discovering $T_2|rep(v)$ for each $v \in T_1$.

To have a linear time algorithm, however, we need to be able to compute $T_2|rep(v)$ quickly. We cite the following result from [18] which will be useful to us in this case.

LEMMA 4.2 (see [18]). *Given a left-to-right ordering of the leaves of a tree and the ability to determine the topology of any triple of leaves a, b, c in constant time, we can construct the tree in linear time.*

To use this lemma we need two things:

(1) we must be able to determine the topology of any triple in T_2 in $O(1)$ time and
(2) we must have for each node in T_1 an ordered representative set, where the ordering is consistent with the left-to-right ordering of the leaves in T_2.

To accomplish (1), we first preprocess T_2 for LCA queries. Then, to determine the topology for the triple a, b, c, we simply compare the LCAs of (a, b), (b, c), and (a, c). The second requirement is more challenging but can also be handled, as we now show.

Computing all ordered representative sets in $O(n)$ time.

- Initially all nodes in T_1 have empty labelings.
- For each $s \in S$, taken in the left-to-right ordering of the leaves in T_2, do the following steps:
  1. trace a path in T_1 from the leaf for s toward the root, until encountering either the root or a node which has already been labeled;
  2. append s to the ordered set for each such node in the path traced (including the first node encountered which has already been labeled).

Figure 2 shows an example of the computation just described.

Note that this computation takes $O(n)$ time since each node v is visited $O(\deg(v))$

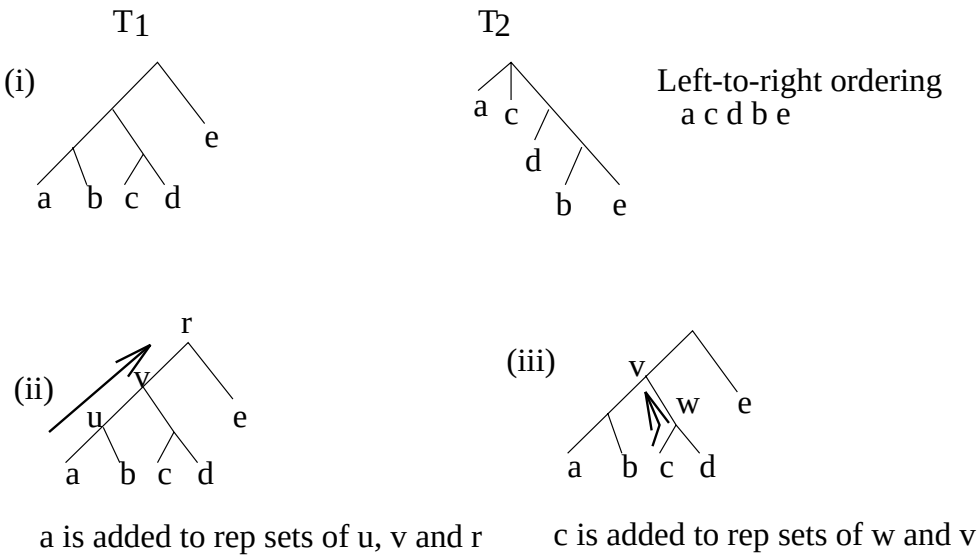


FIG. 2. *Example showing the computation of the representative sets of nodes in T_1 based on the left-to-right ordering of species in T_2.*

times and that the order produced is exactly as required. Thus, for each node $v \in V(T_1)$, we have defined a set of leaves such that each leaf is in a different subtree of v, every subtree of v is represented, and the order in which these leaves appear is the same as the left-to-right ordering in T_2.

We have thus proved Lemma 4.3.

LEMMA 4.3. *We can compute $T_2|rep(u)$ in $O(|rep(u)|)$ time.*

We therefore have the following theorem.

THEOREM 4.4. *Given T_1, T_2, then we can construct a tree T^* such that $C(T^*) = C(T_1) \cup C(T_{olc})$ whenever T_{olc} exists in $O(n)$ time.*

The rest of the task of constructing T_{olc} is in the contraction of unneeded edges.

Contracting T. Now that T^* satisfies $C(T^*) = C(T_1) \cup C(T_{olc})$, we can simply go through each edge in T^* and check if it needs to be kept or must be deleted. Note that edges that were added during the refinement phase are required and do not need to be checked. Therefore, we need only check the original tree edges. Let (u, v) be such an edge with $v = \text{parent}(u)$. From our representative sets for u and v we can easily choose three species a, b, c such that $lca(a, b) = u$ and $lca(b, c) = v$. If the topology of this triple in T_2 is resolved differently than $((a, b), c)$, then we know that edge (u, v) will have to be contracted; if on the other hand $T_2|\{a, b, c\}$ is either (a, b, c) or $((a, b), c)$ then (u, v) will have to be retained in any OLC tree.

OLC Construction Algorithm

Phase 0*: Preprocessing*

Make copies T_1' and T_2' of T_1 and T_2, respectively. For each node v in each tree T_i' $(i = 1, 2)$, compute ordered representative sets ordered by the left-to-right ordering in the other tree. Preprocess each tree T_i' to answer *lca* queries for leaves as well as internal nodes.

Phase I: *Refine* T_1'

Refine T_1' in a postorder fashion so that at the end $C(T_1') = C(T_1) \cup C(T_{olc})$ if T_{olc} exists.

Phase II: *Contract* T_1'

Contract edges $e \in E(T_1')$ such that c_e, the cluster below e, lies in $C(T_1) - C(T_{olc})$.

We have thus shown the following theorem.

Theorem 4.5. *The algorithm stated above constructs the OLC of two trees* T_1 *and* T_2 *if the OLC exists.*

Analysis of Running Time

Phase 0*: Preprocessing*

In [20], Harel and Tarjan give an $O(n)$ time algorithm for preprocessing trees to answer LCA queries in constant time. We have already shown that computing the ordered representative sets takes $O(n)$ time. Thus the preprocessing stage takes $O(n)$ time.

Phase I: *Refining* T_1'

This stage involves local refinements of T_1', and we have shown that the cost of refining around node v is $O(\deg(v))$. Summing over all nodes v we obtain $O(n)$ time.

Phase II: *Contracting edges*

This stage clearly takes only $O(n)$ time.

Theorem 4.6. *Construction of the optimistic local consensus tree can be done in linear time.*

4.1.3. Verification phase. We have identified a candidate optimistic local consensus tree. We now have to decide if this is really such a tree or that no such tree exists.

Lemma 4.4. *Let* T *be a tree on a leaf set* S*. Let* T^* *be obtained from* T *through a sequence of refinements followed by a sequence of edge contractions. Then there exists a function* $f : V(T) \to V(T^*)$ *such that for all* $v \in V(T)$*, there is a subset* S_v *of the children of* $f(v)$ *in* $V(T^*)$ *such that* $\alpha_v = \cup_{v' \in S_v} \alpha_{v'}$*.*

Proof. We define $f(v) = lca_{T^*}(\alpha_v)$. Clearly, $C(T^*) \cup C(T)$ is a compatible set of clusters. Therefore, there is a subset S_v of the children of $f(v)$ such that $\cup_{v' \in S_v} \alpha_{v'} = \alpha_v$. ∎

We take a slight detour and examine the verification of the OLC when the two input trees are both binary. In this case no triple will be unresolved.

Definition 4.8. *A* caterpillar *is a rooted binary tree with only one pair of sibling leaves.*

Given a leaf labeled caterpillar T' with root r and height h, there is a natural ordering induced by T' on its leaves. Let $g : S \to \{1, 2, \ldots, h\}$ be a function where $g(s)$ is the distance of s from r.

Then the species in S can be ordered in the increasing order as $a_1, a_2, \ldots, a_n$, where $a_i \in S$ such that $g(a_1) < g(a_2) \cdots < g(a_{n-1}) \leq g(a_n)$. (Note that the pair of sibling leaves have been arbitrarily ordered.)

DEFINITION 4.9. *Two caterpillars X and Y on the same leaf set are said to be* oppositely oriented *iff for all k, the k smallest elements of X are contained among the $k+1$ largest elements of Y and vice versa. See Figure* 3.

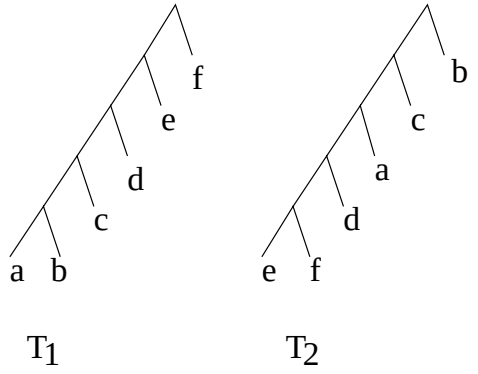


FIG. 3. *Example of oppositely oriented caterpillars.*

PROPOSITION 4.2. *Let T_1 and T_2 be two rooted binary trees on the same leaf set whose OLC is a star. If a, b is a sibling pair of leaves in T_1, then the LCA of a and b in T_2 must be the root of T_2.*

Proof. Suppose Proposition 4.2 is not true. Then there is a species c such that the LCA of (a, c) is above the LCA of (a, b) in T_2. Then $T_1|\{a, b, c\} = T_2|\{a, b, c\}$ and hence the OLC of T_1 and T_2 cannot be a star. □

LEMMA 4.5. *Suppose T_1 and T_2 are binary trees on the same leaf set and suppose that they each have at least five leaves. If their OLC tree is a star, then T_1 and T_2 must be caterpillars.*

Proof. Suppose for contradiction that T_1 is not a caterpillar. Then it has two pairs of sibling leaves (a, b) and (c, d). By the previous proposition each of these pairs must have the root as their LCA in T_2. Thus without loss of generality, a and c lie in the left subtree of the root of T_2, and b and d lie in the right subtree of the root of T_2.

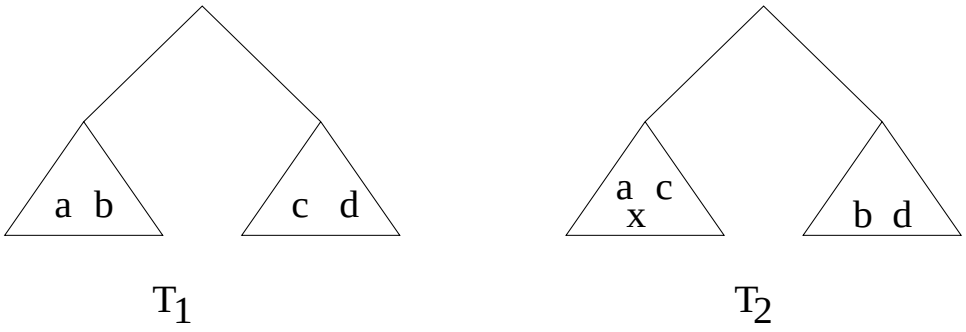


FIG. 4. *Topologies of T_1 and T_2 with respect to a, b, c, d, x.*

Let x be any other species besides a, b, c, and d (see Figure 4). Suppose without loss of generality that x lies in the left subtree of the root of T_2. We will consider the following two triples: x, a, d and x, c, b. In T_2 the topology of these triples will be $((x, a), d)$ and $((x, c), b)$, respectively.

We will show that T_1 agrees on at least one of these triples. There are two cases. If x lies in the left subtree of the root of T_1, then the topology of the triple x, a, d in T_1 is clearly $((x, a), d)$ and if x lies in the right subtree of the root of T_1, then the topology of the triple x, c, b in T_1 is $((x, c), b)$. Thus in either case there is a triple in T_1 which agrees with a triple in T_2, and the OLC cannot be a star. □

LEMMA 4.6. *Let T_1 and T_2 be two caterpillars on the same leaf set. Then the OLC of T_1 and T_2 is a star iff T_1 and T_2 are oppositely oriented caterpillars.*

Proof. Suppose the two caterpillars are oppositely oriented, i.e., they satisfy the two intersection conditions. Let x, y, z be any three leaves and let their indices in the ordering of the leaves of T_1 be $i < j < k$, respectively. Then the topology of x, y, and z in T_1 is $(x, (y, z))$. Looking at the $n - j$ smallest elements in T_2, this set must contain y or z but cannot contain x. Consequently, the topology of the triple in T_2 is not $(x, (y, z))$ and the star is a valid OLC.

Conversely, suppose that the two caterpillars do not satisfy the intersection conditions. Without loss of generality, suppose that there exists at least one k such that the k smallest elements of T_2 are not contained within the $k + 1$ largest elements of T_1. Pick the smallest such k. Say x is the leaf in T_2 with rank k and x does not belong to the set of $k+1$ largest elements of T_1. From the pigeonhole principle, there will exist at least two leaves of T_2 which have ranks greater than k but which are contained in the set of $k + 1$ largest elements of T_1. Suppose the two leaves are y and z. Then $T_1|\{x, y, z\} = T_2|\{x, y, z\} = (x, (y, z))$. This implies that the OLC cannot be a star. □

COROLLARY 4.1. *The OLC for two binary trees can be verified to be a star in linear time.*

Now we return to the general case of verifying the OLC of two trees.

LEMMA 4.7. *Suppose T is the OLC of T_1 and T_2 (on a leaf set S containing at least five species). Then T is a star iff either one of the following holds:*

1. *both T_1 and T_2 are oppositely oriented caterpillars or*
2. *both T_1 and T_2 are stars.*

Proof. The "if" direction is easy to see. We now assume that the OLC, T, is a star. If T_1 contains a triple a, b, c that is unresolved, T_2 must also be unresolved on a, b, c. Conversely whenever T_1 is resolved on a, b, c, T_2 must be (differently) resolved on a, b, c. Thus either both T_1 and T_2 are binary or both are not.

In the case that both T_1 and T_2 are binary, we appeal to the proofs of Lemmas 4.5 and 4.6 to argue that T_1 and T_2 must be oppositely oriented caterpillars.

If T_1 and T_2 are not binary, we will show that for any node v in T_1 with children $\{u_1, \ldots, u_k\}$, $k \geq 3$, there is a node v' in T_2 with children $\{u'_1, \ldots u'_k\}$ such that $\alpha_{u_i} = \alpha_{u'_i}$. Pick any three species a, b, c such that a, b, c is unresolved in T_1 and let $v = lca_{T_1}(a, b, c)$. Then a, b, c must be unresolved in T_2. Let $v' = lca_{T_2}(a, b, c)$. We claim that $\alpha_v = \alpha_{v'}$. To see why, suppose $\alpha_v \neq \alpha_{v'}$ and suppose $x \in \alpha_v, x \notin \alpha_{v'}$ with x being in the same subtree under v as a. Then $T_1|\{b, c, x\} = (b, c, x)$, whereas $T_2|\{b, c, x\} = ((b, c), x)$. This contradicts the assumption that T is a star. Thus $\alpha_v = \alpha_{v'}$.

Next, note that if x and y are under the same child of v in T_1 but under different children of v' in T_2, then there exists a z such that x, y, z is resolved in T_1 but unresolved in T_2. This would contradict the fact the T is a star. This establishes the claim.

This implies that if there is a nonbinary node v that is not the root of T_1, we can find two species a, b ($a \leq v, b \leq v$) and a species c, $c \not\leq v$ such $T_1|\{a, b, c\} = T_2|\{a, b, c\}$.

Thus the root must have three or more children in this case. But this means that if any cluster defined by a child of the root contains two or more species, then there is a triple on which T_1 and T_2 agree. Thus T_1 and T_2 must be stars. □

The verification proceeds as follows:

Phase 0

Suppose the tree constructed by refining T_1 and then contracting the edges in the resulting tree is T. We will do the same modification on T_2, i.e., refine T_2 using the information from T_1 and then contract the edges in the resulting tree as before. Call this tree $T^{'}$. Clearly, if T is not isomorphic to $T^{'}$, we can terminate and output that the OLC does not exist. This is because we know that a compatible set of clusters defines a unique tree and we know that the OLC, if it exists, is uniquely characterized.

Phase 1

If Phase 0 is successful, we then verify further. We compute an ordered representative set for every node w in $V(T)$. For each node w in T, do the following steps.

1. Check if the homeomorphic subtrees of T_1 and T_2 induced by $rep(w)$ are both stars or they are both oppositely oriented caterpillars. If they are neither of these, then terminate and output that the OLC does not exist.
2. Identify the parent of w, say w^*. Look at $rep(w^*)$ excluding the representative element which is below w. Call this set A. Identify the LCAs of $rep(w)$ in T_1 and T_2. Check if there is a species that belongs to A which lies below the LCA of $rep(w)$ in both T_1 or T_2. If so, terminate and output that the OLC does not exist.

Implementation of step 1 *of Phase* 1. Using the left-to-right ordering of the species in T_1, compute the ordered representative set *rep* at each node in T as shown in the previous section. For any $u \in V(T)$, to be able to quickly compute the homeomorphic subtree of T_2 induced by the species in $rep(u)$, we need to know the ordering of theses species as they appear in the left-to-right ordering of T_2. We associate with each u, a new *rep* set, $rep^*(u)$, which is the rearranged version of the species in $rep(u)$ according to their ordering in T_2. We define a function, $limit : S \rightarrow V(T)$, which specifies for each $s \in S$ the node $v \in V(T)$ closest to the root of T such that $s \in rep(v)$. The function *limit* together with the left-to-right ordering of the species in T_2 help in filling the rep^* sets, since s will belong to the rep^* sets of all nodes in the path from s to $limit(s)$. We first show how to compute $limit(s) \forall s \in S$ using algorithm $LIMIT$ and then we show how the rep^* sets are filled.

Initialization:
$limit(s) = +\infty \forall s \in S$.

PROCEDURE LIMIT
For each $v \in V(T)$ visited in a top-down traversal of T,
do {
  Identify $rep(v)$
  For each $s \in rep(v)$ such that $limit(s) = +\infty$
    set $limit(s) = v$
}enddo

Once $limit(s)$ has been identified for all $s \in S$, we proceed to compute $rep^*(u) \forall u \in V(T)$ as follows. Look at the left-to-right ordering of the species in T_2. Now, for each species s in the left-to-right order, we trace a path in T from the leaf for s toward

the root of T and add s to the rep^* set of each node encountered in this path. We terminate when we reach $limit(s)$.

Note that this process of identifying rep and rep^* has to be done only once.

Analysis of running time. The isomorphism test in *Phase* 0 can be performed in $O(n)$ using a simple modification of the tree-isomorphism testing algorithm in [2].

There is an $O(n)$ cost for preprocessing of T_1 and T_2 to answer LCA queries in *Phase* 1.

Our implementation of *step* 1 of *Phase* 1 involves a one-time $O(n)$ cost in preprocessing to identify rep and rep^* for each node in T. Then each time *step* 1 is called on a node $w \in V(T)$, an additional time of $O(\deg(rep(w)))$ is taken.

Exploiting that fact that T_1 and T_2 have been preprocessed to answer LCA queries, it can be seen that each *step* 2 of *Phase* 1 takes $O(\deg(w) + \deg(w^*))$.

Thus the total time taken in the verification phase is $O(n)$.

Correctness of our verification procedure. See Theorem 4.7.

THEOREM 4.7. *If T passes the above tests, then T is the OLC of T_1 and T_2.*

Proof. We need only show that T handles every triple properly. Each of the following cases is handled assuming T has passed the isomorphism test.

Case 1. If T passes the isomorphism test with T', then any triple a, b, c such that the two trees resolve a, b, c differently will be unresolved in T. This follows since T is created by refining and then contracting both T_1 and T_2, and these actions cannot take a resolved triple into a different resolution.

Case 2. This involves a triple a, b, c having the same topology $((a, b), c)$ in both T_1 and T_2. We claim that the first step of *Phase* 1 will pass only if the topology of this triple is $((a, b), c)$. To see why, suppose a, b, c is unresolved in T. (a, b, c cannot be resolved as $(a, (b, c))$ or $((a, c), b)$ in T.) Look at the nodes u and v, which are the LCAs of a, b in T_1 and T_2, respectively. The node w in T, which is the $lca(a, b, c)$, is also $lca(a, b)$ (since a, b, c is unresolved). We infer that $f(u) = w$, where f is the function as defined in Lemma 4.4. This is because any node above w will contain the species c and any node below w will not contain either a or b. By a similar argument, $f(v) = w$. Now, when we look at $rep(w)$ and compute the homeomorphic subtrees of T_1 and T_2 induced by $rep(w)$, in both of these induced trees, there will exist three species x, y, z such that x, y are both below u (and v) in T_1 (and T_2) and z is not in the character defined by u (and v). Thus in both the induced trees, the triple x, y, z will have the same topology $((x, y), z)$. That is, these induced trees will neither be both stars nor both oppositely oriented caterpillars. Thus the verification process will terminate and output that the OLC does not exist.

Case 3. This involves a triple a, b, c which is resolved as $((a, b), c)$ in one tree and unresolved in the other. The proof of this case essentially follows the lines of the proof of Case 2.

Case 4. This involves a triple a, b, c which is unresolved in both trees. We claim that the second step of *Phase* 1 will pass only if this triple is unresolved in T. To see why, suppose a, b, c is resolved as $((a, b), c)$ in T. Let $lca_T(a, b, c) = x$ and let $lca_T(a, b) = y$ and also suppose without loss of generality that x is the parent of y. Let y_1 be the child of y such that $a \in \alpha_{y_1}$ and let y_2 be the child of y such that $b \in \alpha_{y_2}$. Let $z \neq y$ be the child of x such that $c \in \alpha_z$.

Let $u = lca_{T_1}(a, b, c)$ and $v = lca_{T_2}(a, b, c)$.

We will look at functions f_1 and f_2 defined by Lemma 4.4 from $V(T)$ to $V(T_1)$ and $V(T_2)$, respectively. Clearly $f_1(y) = u$ and $f_2(y) = v$. Note that the cluster defined by any child of u can have a nonempty intersection with at most one of α_{y_1}

and α_{y_2}. This is similar for v. Thus any representatives chosen from α_{y_1} and α_{y_2}, respectively, have their LCA at u in T_1 and at v in T_2. However, $f_1(z) \leq_{T_1} u$ and $f_2(z) \leq_{T_2} v$. Thus any representative chosen from α_z will lie below u and v in T_1 and T_2, respectively, causing us to conclude that the OLC does not exist. □

4.2. PLC. Recall the definition of the PLC tree: *Let T_1 and T_2 be two rooted trees on the same leaf set S. A rooted tree T is called the* PLC *of T_1 and T_2 iff for each triple a, b, c, $T|\{a, b, c\} = ((a, b), c)$ iff $T_1|\{a, b, c\} = T_2|\{a, b, c\} = ((a, b), c)$.*

Just like the OLC, the PLC tree need not always exist either.

4.2.1. Characterization. The following theorem characterizes the PLC tree of two trees T_1 and T_2.

THEOREM 4.8. *Let T_1 and T_2 be two trees on the same leaf set S. If the PLC tree T_{plc} of T_1 and T_2 exists, then it is identically equal to T, where $C(T) = C(T_1) \cap C(T_2)$.*

Proof. Pick any cluster $\alpha \in C(T)$. Since α belongs to both the trees, if we look at any triple x, y, z with $x, y \in \alpha$ and $z \notin \alpha$, then this triple will have to be resolved as $((x, y), z)$. Thus $\alpha \in C(T_{plc})$.

Conversely, pick any cluster $\alpha \notin C(T)$. We have two subcases here.

1. α is not compatible with at least one of $C(T_1)$ or $C(T_2)$. In this case, from Theorem 4.1, $\alpha \notin C(T_{plc})$.
2. α is compatible with both $C(T_1)$ and $C(T_2)$. In this case, pick those nodes from T_1 and T_2 that define the smallest clusters containing α. We can pick a triple a, b, c such that $a \in \alpha, b \in \alpha, c \notin \alpha$ and this triple is unresolved in either T_1 or T_2. Thus $\alpha \notin C(T_{plc})$. □

4.3. Construction phase. By Theorem 4.8, the PLC tree, if it exists, is identically the strict consensus tree. Thus to construct the PLC tree, it suffices to use the $O(n)$ algorithm in [9] for the strict consensus tree.

4.3.1. Verification phase. Let T_1 and T_2 be the input trees, and let T be the strict consensus tree constructed using the algorithm in [9]. We want to be able to verify whether T is actually the PLC in the case that T is a star. If T_1 or T_2 is already a star then there is nothing to verify since T is the true PLC. So assume that this is not the case.

There are two cases which we will consider. The first is when either of T_1 or T_2 (say T_1) has at least two children of the root which are not leaves. The second case is when both T_1 and T_2 have exactly one child of the root which is not a leaf. Having made observations about these cases, we can apply a divide and conquer strategy as seen by the following lemma.

LEMMA 4.8. *Let T_1 and T_2 be rooted trees on the same leaf set and let α be a cluster in their intersection. Let T be the strict consensus tree of T_1 and T_2. Let e_1, e_2, and e be the edges in T_1, T_2, and T respectively, that are above the respective internal nodes which define the cluster α. Let a be a species in α. Then T is a PLC for T_1 and T_2 iff*

(1) *the subtree below e is a PLC for the subtrees below e_1 and e_2, and*
(2) *upon replacing the subtrees below e, e_1, and e_2 by a in T, T_1, and T_2, respectively, T is a PLC for T_1 and T_2.*

Proof. Clearly, if T is the PLC tree for T_1 and T_2 then conditions (1) and (2) will hold. Conversely, if (1) and (2) hold, but T is not the PLC tree for T_1 and T_2, then there is some triple a, b, c such that T incorrectly handles this triple. If all of a, b, c are below e then by condition (1), T handles a, b, c correctly. Similarly if at least two are above e, then by condition (2), T handles this triple correctly. It remains to show

that T handles all triples where exactly two of a, b, c are below and one is above the edge e. But then, since the cluster $\alpha \in C(T_1) \cap C(T_2) = C(T)$, in each of T_1, T_2, and T, we have $((a, b)c)$, so that T handles this triple properly. Thus T is a PLC for T_1 and T_2. □

Thus the verification proceeds by traversing T in a postorder fashion and at the end of each successful verification step replacing the subtree by a single element belonging to the cluster defined by the root of the subtree. We now discuss the details of each verification step.

LEMMA 4.9. *Suppose T_1 and T_2 are two trees on the same leaf set S with T_1 having at least two children of the root which are not leaves. Let $\alpha_1, \ldots, \alpha_l$ be the maximal clusters of T_1 and $\beta_1, \ldots, \beta_m$ be the maximal clusters of T_2. Then T, their PLC, is a star iff $\forall i, j\ |\alpha_i \cap \beta_j| \leq 1$.*

Proof. Suppose $\forall i, j\ |\alpha_i \cap \beta_j| \leq 1$. This means that $\forall x, y$, if $lca(x, y)$ in T_1 is below the root, then in T_2, $lca(x, y)$ is the root. Thus for any triple x, y, z, their topologies in T_1 and T_2 do not agree. Thus T is a star.

Suppose $\exists i, j\ |\alpha_i \cap \beta_j| > 1$. Thus α_i is defined by a node which is not a leaf. Look at an α_k, $k \neq i$, such that the node in T_1 defining α_k is not a leaf node. There are two cases to handle here. Either at least one species in α_k is not in β_j or all species in α_k are in β_j (i.e., $\alpha_k \subset \beta_j$).

In the former case, pick that species z that is in α_k but not in β_j. Also pick those two species x, y that are in $\alpha_i \cap \beta_j$. Both T_1 and T_2 agree on the triple x, y, z; namely this triple has topology $((x, y), z)$ in both the trees. Thus T cannot be a star.

In the latter case, since we know that $\beta_j \neq S$, we can pick two species x, y from α_k and another species z from $S - \beta_j$. In both T_1 and T_2, the topology of this triple is $((x, y), z)$. Thus T cannot be a star. □

Since each species belongs to at most one of these maximal clusters in each tree, this test can be done in linear time.

The following lemma handles the case when both T_1 and T_2 have exactly one child of the root which is not a leaf.

LEMMA 4.10. *Suppose T_1 and T_2 are two trees on the same leaf set S and T and their PLC is a star. Suppose both T_1 and T_2 have exactly one child of the root each which is not a leaf. Let $s_1, \ldots, s_k$ be leaves in T_1 which are children of the root. Let v be the LCA in T_2 of $s_1, \ldots, s_k$. Then every child of v contains at most one species $x \in S - \{s_1, \ldots, s_k\}$. Moreover, for any pair of species $x, y \in S - \{s_1, \ldots, s_k\}$, the LCA of x and y in T_2 lies on the path from v to the root.*

Proof. Suppose $\exists$ a child of v which contains at least two species from $S - \{s_1, \ldots, s_k\}$. Then by picking x, y such that they both lie under this child if v in T_2 and picking an s_i out of $s_1, \ldots s_k$ that lies under a different child of v, we find that both trees have the same topology for the triple x, y, s_i. Thus T cannot be a star. Furthermore, if $\exists x, y \in S - \{s_1, \ldots, s_k\}$ such that $lca(x, y)$ in T_2 does not lie on the path from v to the root, then the triple x, y, s_1 would have identical topologies in both trees and T wouldn't be a star. □

DEFINITION 4.10. *A rooted tree T is a* millipede *if the set of internal nodes of T defines a single path from the root to a leaf. See Figure* 5.

Let $S_1 = S - \{s_1, s_2, \ldots, s_k\}$. We have that $T_2|S_1$ is a millipede (say, T_2^*).

Let $u_1, \ldots, u_l$ be the children of the root in T_2^*, which are leaves. Look at $T_1|S_1$ (say, T_1^*). Either, T_1^* has one nonleaf child or it has at least two nonleaf children. In the former case, we can apply the previous lemma and infer that $T_1^*|(S_1 - \{u_1, \ldots, u_l\})$ will be a millipede. In the later case, we can apply Lemma 4.10 to check if the PLC is a star.

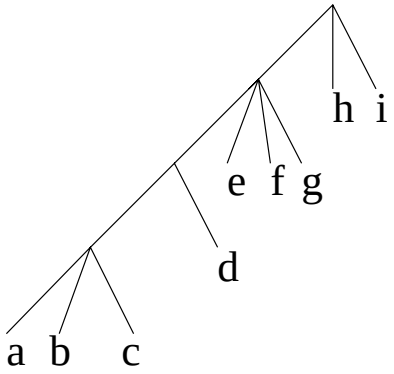


FIG. 5. *An example of a millipede.*

In the following subsection we will show how to verify if T is a star when both the input trees are millipedes.

4.3.2. Verification when both the input trees are millipedes. The proof of the following lemma is straightforward.

LEMMA 4.11. *Suppose T_1 and T_2 are two millipedes on the same leaf set S. Then their PLC T is a star iff there exists no triple such that both trees have the same resolved topologies on the triple.*

We now describe a linear time algorithm for verifying that T_1 and T_2 have no triple on which they have the same topology.

We define an ordering on the species in T_1 using the function $f : S \to \{1, \ldots, h\}$, where $f(s) =$ distance of s from the root of T_1 and h is the height of T_1.

In T_2, we can write S as the union of all the sets in the sequence $S_1, S_2, \ldots, S_k$, where k is the height of T_2 and each S_i contains exactly those species which are at a distance i from the root of T_2. Now, in each S_i replace each species s in this set with $f(s)$. Call this multiset of integers M_i. We thus get a sequence $M_1, M_2, \ldots, M_k$ of multisets.

DEFINITION 4.11. *We will say a triple of integers p, q, r is* special *if*

- $p < q$, $p < r$;
- $p \in M_{j_1}, q \in M_{j_2}, r \in M_{j_3}$, *with* $1 \leq j_1 < j_2 \leq k$ *and* $1 \leq j_1 < j_3 \leq k$.

We observe that the PLC of T_1 and T_2 is a star iff no special triple p, q, and r exists.

The following *CHECK_PLC* algorithm takes as input the sequence $M_1, M_2, \ldots, M_k$ and returns $FAIL$ if there exists a special triple of integers, and otherwise it returns $PASS$.

CHECK_PLC works by scanning the multiset M_i in the ith iteration. It makes use of three variables *global_min*, *local_min*, and *temp*. At the start of the ith iteration, *global_min* stores the smallest integer seen in the first $i - 1$ multisets. The variable *local_min* is used to store the smallest integer a such that $\exists b$ for which $a < b$ and $a \in M_j, b \in M_l$ with $1 \leq j < l < i$. (*local_min* is initialized to $+\infty$.) The variable *temp* is initialized to 0. As long as *temp* remains 0, *local_min* $= +\infty$. If *temp* is nonzero, then *local_min* stores a and *temp* stores some b for which the previously mentioned relationship between a and b holds. At the ith iteration, *CHECK_PLC* either returns $FAIL$ (if a special triple exists) or, if necessary, it modifies the variables *global_min*, *local_min*, and *temp* to hold their intended values for the first i multisets of the sequence.

The reasoning for storing these values at the start of the ith iteration is as follows. If $\exists p$ in some M_j, and $q, r \in M_i$ $(1 \leq j < i)$ such that p, q, r is a special triple, then *global_min* together with $q, r \in M_i$ are also a special triple since *global_min* $\leq p$. Similarly, if $\exists p$ in some M_j, $q \in M_l$, $r \in M_i$ $(1 \leq j < l < i)$, such that p, q, r is a special triple, then *local_min*, *temp*, and $r \in M_i$ are also a special triple.

We now describe *CHECK_PLC*.

Initialization:

$global\_min = Min(M_1)$
$local\_min = +\infty$
$temp = 0$.

The procedure outputs $FAIL$ (and terminates) if the PLC is not a star; it outputs $PASS$ otherwise.

PROCEDURE CHECK_PLC
For $2 \leq i \leq k$,
do {
If $temp = 0$, then *Step* 1, else *Step* 2.
Step 1
do {
Scan through M_i;
Identify $A = \{y | y \in M_i, global\_min < y\}$;
If $|A| \geq 2$, then output $FAIL$;
If $|A| = 1$, then set $temp = y$, where $y \in A$
$local\_min = global\_min$
$global\_min = Min\{global\_min, Min(M_i)\}$;
If $|A| = 0$, then set $global\_min = Min(M_i)$.
} enddo
Step 2
do {
Scan through M_i;
Identify $A = \{y | y \in M_i, global\_min < y\}$;
Identify $B = \{z | z \in M_i, local\_min < z\}$;
If either $|A| \geq 2$ or $|B| \geq 1$, then output $FAIL$;
Else
If $|A| = 1$ then
If $global\_min < Min(M_i)$, then set $local\_min = global\_min$
$temp = Min(M_i)$;
If $global\_min > Min(M_i)$, then set $local\_min = global\_min$
$temp = y$, where $y \in A$
$global\_min = Min(M_i)$;
If $|A| = 0$ then set $global\_min = Min(M_i)$.
} enddo
} enddo
Output $PASS$

Analysis of running time. *CHECK_PLC* runs in linear time since each M_i is scanned only a constant number of times.

THEOREM 4.9. *Algorithm* CHECK_PLC *is correct.*

Proof. By induction, observe that *Step* 1 is executed at the ith iteration if $\forall j, l, x$, where $1 \leq j < l < i$ and $x \in M_l$, $Min(M_j) \geq x$. It then follows that if *Step* 1 is executed at the ith iteration, then at the start of that iteration $temp = 0$,

$global\_min = Min(M_{i-1})$, and $local\_min = +\infty$. Thus, in this case $global\_min$ stores the smallest integer seen in the first $i-1$ multisets. Now, in the first i multisets, if any special triple p, q, r exists such that $p \in M_j$ $(j < i)$ and $q, r \in M_i$, then $CHECK\_PLC$ correctly outputs $FAIL$ since $global\_min \leq p$. Otherwise we have two cases, depending upon the value of A. If $|A| = 1$, then the variables $global\_min$, $temp$, and $local\_min$ are updated so that $global\_min$ holds the smallest value in the first i multisets. Also, $local\_min$ now correctly holds the smallest value a for which there exists a b (stored in $temp$) for which $a < b$ and $a \in M_j, b \in M_l$ with $1 \leq j < l < i$. In the other case $|A| = 0$, in which case $global\_min$ is updated to hold $Min(M_i)$ (which is the smallest value in the first i multisets).

Observe that once $temp$ is updated to store a nonzero value, it never stores a 0 again. Thus, once $temp$ is set to a nonzero value in iteration i', then from iteration $i'+1$ to iteration k, *Step* 2 is executed.

Assume that *Step* 2 is executed in some iteration i' and assume, inductively, that at the start of iteration i', $global\_min$ stores the smallest value in the first $i'-1$ multisets and $local\_min$ stores the smallest value a for which there exists a b (stored in $temp$) such that $a < b$ and $a \in M_j, b \in M_l$ with $1 \leq j < l < i'$. Then in iteration i', it can be easily seen that $CHECK\_PLC$ correctly outputs $FAIL$ if there exist a special triple p, q, r such that $p \in M_{i_1}, q \in M_{i_2}$ $(i_1 < i_2 < i')$, $r \in M_{i'}$ or $p \in M_{i_1}, q, r \in M_{i'}$ $(i_1 < i')$. Otherwise, for both the cases when $|A| = 1$ and $|A| = 0$, *Step* 2 ensures that after iteration i', $global\_min$ stores the smallest value in the first i' multisets and $local\_min$ stores the smallest value a for which there exists a b (stored in $temp$) such that $a < b$ and $a \in M_j, b \in M_l$ with $1 \leq j < l \leq i'$.

Using the above arguments, it can be seen that $CHECK\_PLC$ gives the correct output on any sequence of multisets. □

Thus we also have the following theorem.

THEOREM 4.10. *Given two millipedes T_1 and T_2, we can check if their PLC is a star in linear time.*

4.4. Summary. We have used three general techniques in constructing local consensus trees for these two total local consensus rules:

- we characterize the local consensus tree (that is, we define the set $C(T)$ of binary characters which encode the consensus tree T);
- we use the character encoding of the consensus tree if possible to construct the tree efficiently; and
- we verify that the constructed tree is the local consensus tree.

Some comments about the construction phase are in order. When working with conservative local consensus functions, assuming the local consensus tree T exists, it is possible to construct the local consensus tree T in two phases: a refinement phase in which one of the input trees T_i is refined to produce a tree T^* satisfying $C(T^*) = C(T_i) \cup C(T)$ and then edges are contracted in T^* to produce a tree T^{**} such that $C(T^{**}) = C(T)$.

5. Optimization problems.

5.1. Introduction. The local consensus rules we have seen so far are such that the output tree satisfying the constraints of a particular local consensus rule need not exist. Yet characterizing these rules and developing fast algorithms for them are important because if the consensus tree exists, then we can say something very concrete about it. The nonexistence of the consensus tree in all cases does motivate the need to look at the *optimization versions* of local consensus, where solutions

always exist. We will now describe some natural optimization problems for local consensus tree construction. In these problems, which we call *relaxed* versions, we will consider certain constraints to be absolutely required and let others be desirable but not required. Then we seek a tree meeting all the required constraints and as many of the desirable constraints as possible. We now define some obvious relaxed versions but note that many other versions are equally desirable.

Recall our discussion in section 3.3 regarding a profile being constant, compatible, and incompatible on a triple. The first optimization problem we consider is where we insist that all triples on which the profile is incompatible or is unresolved and constant are left unresolved, and then we seek to leave as resolved a maximal set of triples on which the profile is constant and resolved. This is relaxed version I (RV-I). The second problem is where we insist that all the triples, which the profile leaves as resolved and constant, be left resolved the same, and then we seek to leave a maximal set of the remaining triples as unresolved in the consensus tree. This is relaxed version II (RV-II). The third problem is where we insist that all triples on which the profile is incompatible or leaves unresolved and constant are left unresolved, and we seek to leave as resolved a maximal set of triples on which the profile is constant and resolved or is compatible. This is RV-III. In addition, RV-III also insists that all the resolved triples in the consensus tree be compatible with the profile. Finally, we look at an interesting rule LCR1, where we insist that all triples be left resolved on which the profile is constant and resolved or is compatible. This tries to capture the optimistic features of the OLC model. Unfortunately, the consensus tree need not always exist. We give a counterexample to show this.

5.2. Specific relaxed versions.

DEFINITION 5.1. *Let T_1 and T_2 be two rooted trees (not necessarily binary) on the same leaf set S. A rooted tree T is called an* RV-I *of T_1 and T_2 if whenever a triple a, b, c has differing topologies on T_1 and T_2, or both T_1 and T_2 leave a, b, c as unresolved, then that triple is unresolved in T and in addition T preserves the topology of a maximal set of triples which are resolved identically in T_1 and T_2.*

To prove the existence of an RV-I tree it is sufficient to show that there exists a tree where every triple on which T_1 and T_2 disagree is unresolved. The set of trees with this property can be partially ordered based on the set of triples (on which T_1 and T_2 agree) whose topology they preserve. Once this partial order is known to be nonempty, we have proved the existence of an RV-I since any maximal element in this partial order is such a consensus tree.

We note that if T has the star topology it leaves unresolved all triples on which T_1 and T_2 disagree. Hence the partial order is nonempty and the RV-I tree always exists. In section 5.3 we show that this tree is unique.

DEFINITION 5.2. *Let T_1 and T_2 be two rooted trees (not necessarily binary) on the same leaf set S. A rooted tree T is called an* RV-II *of T_1 and T_2 if T preserves the topology of all triples which are resolved identically in T_1 and T_2. In addition, T should leave unresolved a maximal set of triples on which T_1 and T_2 disagree or which are unresolved in both T_1 and T_2.*

Using an argument similar to the one used to prove the existence of an RV-I tree and noting that T_1 (or T_2) itself preserves the topology of all triples on which T_1 and T_2 agree, we conclude that the RV-II always exists. In section 5.4 we give an algorithm to construct the RV-II tree.

DEFINITION 5.3. *Let T_1 and T_2 be two rooted trees on the same leaf set S. Let T be a rooted tree on the same leaf set. Consider the following rules.*

Rule 1a. *If a triple a, b, c is resolved as $((a, b), c)$ in one tree and as $(a, (b, c))$ in the other, we require that it be unresolved.*

Rule 1b. *If a triple a, b, c is unresolved in both the trees, then we require that it be unresolved.*

Rule 2. *If a triple a, b, c is resolved as $((a, b), c)$ in one tree and is either resolved as $((a, b), c)$ or unresolved in the other tree, then we require it to be resolved as $((a, b), c)$.*

The tree T is called the relaxed version III (RV-III) *of T_1 and T_2 if*

1. *it always satisfies Rules* 1a *and* 1b *for triples;*
2. *it also satisfies Rule* 2 *for a maximal number of triples;*
3. *if a triple a, b, c is resolved as $((a, b), c)$ in T, then it is not resolved as $(a, (b, c))$ or $((a, c), b)$ in either T_1 or T_2.*

In section 5.5 we will show that an RV-III tree also always exists and is unique.

In the next subsections, we will look at the different relaxed versions in greater detail.

5.3. RV-I. In this subsection we will show that the RV-I of two rooted trees T_1 and T_2 is actually the strict consensus of these two trees.

THEOREM 5.1. *If T_1 and T_2 are two rooted trees, then their RV-*I *tree T always exists and is identically the strict consensus of T_1 and T_2.*

Proof. The existence of the RV-I tree T, was shown in section 5.2. Now we show that this tree is the strict consensus tree. Suppose there exists a triple a, b, c resolved differently in T_1 and T_2 as, say, $((a, b), c)$ and $(a, (b, c))$ (or (a, b, c)), respectively. Say the $lca_{T_1}(a, b) = u$ and $lca_{T_2}(b, c) = v$. Clearly, neither α_u nor α_v is in the strict consensus tree. Thus the strict consensus tree leaves unresolved any triple that has different topologies in T_1 and T_2.

Let T' be a tree in which for every triple a, b, c on which T_1 and T_2 differ, T' has an unresolved topology on this triple. Now suppose it is possible that T' contains a cluster that is not in $C(T_1) \cap C(T_2)$. Let α be this cluster and suppose without loss of generality that α is not a cluster of T_1. In T', for any pair of species $x, y \in \alpha$ and species $z \notin \alpha$ the topology has to be $((x, y), z)$. However, if this is also the case in T_1, then T_1 must also possess the cluster α contradicting our assumption. Thus there must exist a pair of species $x, y \in \alpha$ and a species $z \notin \alpha$ such that in T_1 their topology is not $((x, y), z)$. But this implies that T' cannot be an RV-I. Hence any candidate T' for an RV-I can only contain the clusters in the intersection of the cluster sets of T_1 and T_2.

If T' contains a proper subset of the clusters in the intersection of the sets of clusters of T_1 and T_2, then there exists a triple a, b, c on which T' has an unresolved topology while the strict consensus tree has a resolved topology that agrees with the topologies of T_1 and T_2. Hence the strict consensus of T_1 and T_2 is the RV-I tree of T_1 and T_2. □

As a consequence, the RV-I can be constructed in $O(n)$ time using the algorithm in [9], and there is no need to verify that the tree constructed is correct.

5.4. RV-II. In the RV-II problem we require that any triple on which the trees T_1 and T_2 agree must have its topology preserved in the consensus tree T. Further T should leave unresolved a maximal set of triples on which T_1 and T_2 disagree or both leave unresolved.

Previously we showed that the RV-II exists. We note that the RV-II tree is not unique. The construction of the RV-II can be accomplished by defining the set $A = \{((a, b), c) : T_1|\{a, b, c\} = T_2|\{a, b, c\} = ((a, b)c)\}$. This set of rooted triples can then be passed to the algorithm of Aho et al. [3], which computes a tree (if it exists)

having the required form on every triple in the set and also leaving a maximal set of additional triples outside that set unresolved. The algorithm in [3] takes $O(pn)$ time where $p = |A|$. Recall the proof of Theorem 3.1 for a description of the algorithm. Since in our case $p \in O(n^3)$, the use of the algorithm of [3] would result in a running time of $O(n^4)$. We will obtain a speedup to an $O(n^2)$ algorithm (which includes the verification) for the construction of the RV-II tree by using the fact that the tree necessarily exists.

5.4.1. An improved algorithm for RV-II. We will now describe an $O(n^2)$ time algorithm to construct an RV-II tree. We start by making a few observations about the RV-II tree T constructed by the algorithm of [3].

We will use α's to denote the clusters in T_1 and β's to denote the clusters in T_2. Suppose α and β are maximal clusters in T_1 and T_2, respectively, and suppose $\alpha \cup \beta \neq S$. Then we claim that $\alpha \cap \beta$ (if nonempty) will be a maximal cluster in T. This is because $\exists a \in S-(\alpha \cap \beta)$ such that $\forall x, y \in (\alpha \cap \beta)$, $T_1|\{x, y, a\} = T_2|\{x, y, a\} = ((x, y), a)$ and thus the elements of $(\alpha \cap \beta)$ all belong to one component of the graph which is constructed in the execution of the algorithm of [3]. Furthermore, $(\alpha \cap \beta)$ is exactly equal to one component of this graph since the algorithm never adds an edge between two nodes in the graph unless it is forced to and it can be seen that no element x in $(\alpha \cap \beta)$ is such that $\exists y, a \in S-(\alpha \cap \beta)$ with $T_1|\{x, y, a\} = T_2|\{x, y, a\} = ((x, y), a)$.

Thus, if $\alpha \cup \beta \neq S$, then $\alpha \cap \beta$ (if nonempty) is a maximal cluster in T. The case where $\alpha \cup \beta = S$, $\alpha \cap \beta \neq \emptyset$, can occur for at most one child of the root of T_1 and one child of the root of T_2 as the following lemma shows.

LEMMA 5.1. *Let T_1 and T_2 be two trees on the same leaf set S. Let $\alpha_1, \ldots, \alpha_k$ be the maximal clusters of T_1 and $\beta_1, \ldots, \beta_l$ be the maximal clusters of T_2. Then the case where $\alpha_i \cup \beta_j = S$, $\alpha_i \cap \beta_j \neq \emptyset$ can occur for at most one i and one j.*

Proof. Suppose not. Let $\alpha_i \cup \beta_j = S$, $\alpha_i \cap \beta_j \neq \emptyset$, $\alpha_{i^*} \cup \beta_{j^*} = S$, and $\alpha_{i^*} \cap \beta_{j^*} \neq \emptyset$, perforce with $i \neq i^*$ and $j \neq j^*$. Since $\alpha_i \cap \alpha_{i^*} = \emptyset$, we have that $\alpha_i \subseteq \beta_{j^*}$. But since $\alpha_i \cap \beta_j \neq \emptyset$, this implies that $\beta_j \cap \beta_{j^*} \neq \emptyset$. This is a contradiction since β_j and β_{j^*} are clusters defined by the children of the root and hence should be disjoint. □

Recall that the maximal clusters form a partition of the species set S (in each of T_1, T_2, and T). Also, from the above discussions we have that (i) $\alpha \cup \beta \neq S$ implies that $\alpha \cap \beta$ is a maximal cluster in T and (ii) there can be at most one case for which $\alpha \cup \beta = S$. These observations imply that in the case when $\alpha \cup \beta = S$, then $\alpha \cap \beta$ is the union of some maximal clusters of T.

With the above characterization a high-level description of the algorithm to construct T can be given as follows.

RV-II CONSTRUCTION ALGORITHM

1. For each pair of maximal clusters $\alpha \in C(T_1)$ and $\beta \in C(T_2)$ such that $\alpha \cap \beta \neq \emptyset$ and $\alpha \cup \beta \neq S$, recursively compute the tree on $\alpha \cap \beta$ and make its root a child of the root of T.
2. If there are maximal clusters α and β such that $\alpha \cup \beta = S$ but $\alpha \cap \beta \neq \emptyset$, compute the partition of $\alpha \cap \beta$; recursively compute the tree for each component of the partition and make the roots of these trees children of the root of T.

Computing the partition of $\alpha \cap \beta$ in step 2 is described together with the implementation details.

Implementation details and running time analysis. Note that this algorithm does not require an explicit verification of the constructed tree, since in fact we know that the tree exists and we are simply computing it by mimicking efficiently what the algorithm in [3] would create.

There are at most n recursive stages. We will show that each stage can be implemented in $O(n)$ time thereby proving the $O(n^2)$ bound.

To handle case 1 it is important not to waste time on empty intersections. So we consider each species in turn and label the intersection in which that this species lies. Thus we will identify at most n nonempty intersections. Let $\alpha \cap \beta$ be one such intersection. To recurse, we need to find homeomorphic subtrees of T_1 and T_2 that have $\alpha \cap \beta$ as the leaf set. We will show how to do this in time proportional to the number of leaves in $\alpha \cap \beta$.

Assume that T_1 and T_2 have been preprocessed for LCA queries. Also note that we know the left-to-right ordering of all leaves of T_1 as well as of T_2. Given the leaves in $\alpha \cap \beta$, their left-to-right ordering is also known and is the one induced by the overall left-to-right ordering. By Lemma 4.2 we can reconstruct the topology of the tree in linear time.

Thus case 1 can be handled in $O(n)$ time.

We now describe how to handle case 2 also in $O(n)$ time. We will construct a graph $G = (V, E)$ such that $V(G) = \alpha \cap \beta$. The edges will be added so that, finally, each component in G corresponds to a maximal cluster in the RV-II tree.

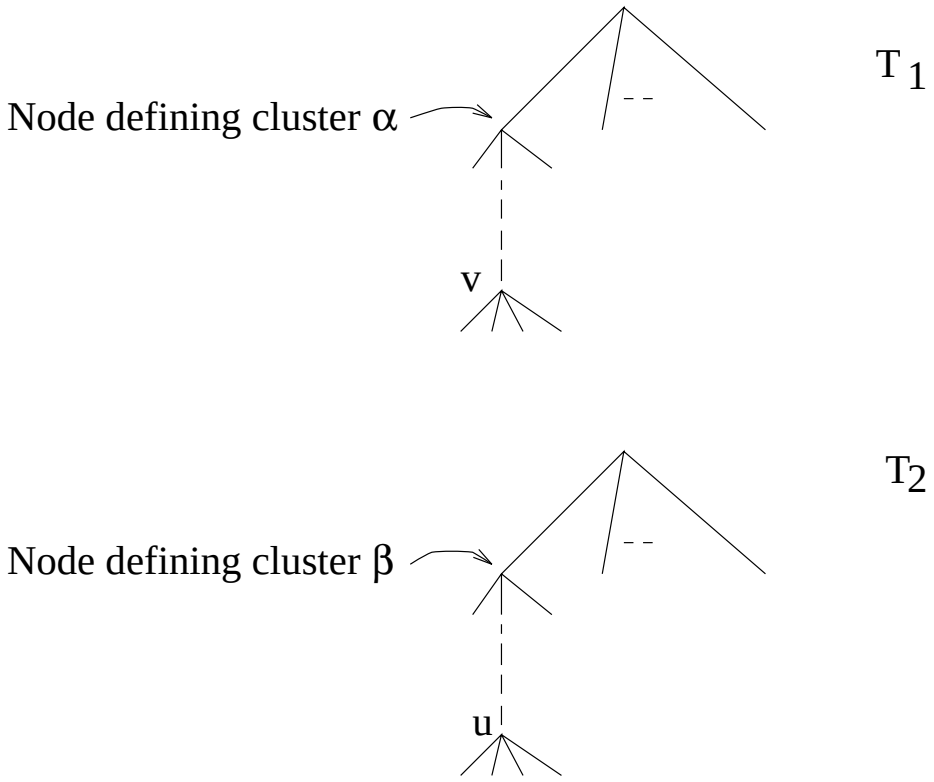


FIG. 6. *Figure showing nodes v and u.*

Identify the LCA, say, u, of the species in $S - \alpha$ in T_2 and similarly the LCA, say, v, of the species in $S - \beta$ in T_1. In T_2, u will be a descendent of the node defining β, and in T_1, v will be a descendent of the node defining α. See Figure 6. In T_1 let v_1 through v_p be the nodes in the path from the root to v, where $v_1 = root$ and $v_p = v$. Similarly, in T_2, let u_1 through u_q be the nodes in the path from the root to u, where

$u_1 = root$ and $u_q = u$. We will say that δ is a *special cluster* if for some v_i, $1 \le i \le p$ (or some u_j, $1 \le j \le q$), δ is a cluster defined by a child of v_i (or u_j) that is not on the path from the root to v (or u).

Let $\delta_1, \ldots, \delta_l$ be the special clusters in T_1 and let $\gamma_1, \ldots, \gamma_m$ be the special clusters in T_2. A pair of species $x, y \in (\alpha \cap \beta)$ will be in the same component of the graph G if $\exists z$ such that $T_1|\{x, y, z\} = T_2|\{x, y, z\} = ((x, y), z)$. There are two cases depending on whether $z \in (\alpha \cap \beta)$ or not. We will now describe how to handle these two cases: Cases 2a and 2b.

Case 2a [$z \notin (\alpha \cap \beta)$]. In this case, it suffices to look at all $\alpha \cap \gamma_i$ and $\beta \cap \delta_j$, and for each intersection put its elements in the same component of G. This is evident from the following lemma.

LEMMA 5.2. *Let α, β be maximal clusters of T_1 and T_2, respectively, such that $\alpha \cup \beta = S$ and let $x, y \in (\alpha \cap \beta)$. Then $\exists z \in S - (\alpha \cap \beta)$ such that $T_1|\{x, y, z\} = T_2|\{x, y, z\} = ((x, y), z)$ iff both x and y belong to some $\alpha \cap \gamma_i$ or $\beta \cap \delta_j$.*

Proof. Suppose $\exists z \in S - (\alpha \cap \beta)$ such that $T_1|\{x, y, z\} = T_2|\{x, y, z\} = ((x, y), z)$. Since $\alpha \cup \beta = S$, the only cases we have to consider are when z is in exactly one of α or β. So suppose $z \in \alpha, z \in S - \beta$ (the other case can be handled similarly). Then z belongs to a special cluster δ_i, which is defined by some child of the node v in T_1. (Recall that node v is the LCA of $S - \beta$ in T_1.) Since $T_1|\{x, y, z\} = ((x, y), z)$, we have that either both x, y belong to δ_i or neither belongs to δ_i. If both $x, y \in \delta_i$, then clearly $x, y \in (\beta \cap \delta_i)$. For the case when neither x nor y is in δ_i, we can conclude that both x, y are in some special cluster δ_j (since $T_1|\{x, y, z\} = ((x, y), z)$). Thus we have that $x, y \in (\beta \cap \delta_j)$.

Suppose x, y belong to some $\alpha \cap \gamma_i$ or $\beta \cap \delta_j$; specifically, say x, y belong to some $\beta \cap \delta_j$. There are two cases to handle. The first case is if the node v' defining the special cluster is not a child of the node v. In this case, we can pick a species $z \in S - \beta$ such that $T_1|\{x, y, z\} = T_2|\{x, y, z\} = ((x, y), z)$. The second case is when the node v' is a child of the node v. In this case, pick a species $z \in S - \beta$ from the special cluster which is defined by a node v'' (where $v' \ne v''$) and v'' is below v. We have that $T_1|\{x, y, z\} = T_2|\{x, y, z\} = ((x, y), z)$. Thus in both the cases we have that there exists such a z with $z \in S - (\alpha \cap \beta)$. □

Thus, for each i, connect all vertices in $\alpha \cap \gamma_i$ (in G) by a path and do the same for each j and the vertices in $\beta \cap \delta_j$. Note that this can be done in $O(n)$ by using the same idea as in Case 1.

Case 2b [$z \in (\alpha \cap \beta)$]. Note that we are only interested in identifying x, y such that $lca(x, y)$ in T_1 is a node that is on the path from the root of T_1 to the node v, and the $lca(x, y)$ in T_2 is a node that is on the path from the root of T_2 to the node u. To see why, if, say, $lca_{T_1}(x, y) \ne v_i \forall 1 \le i \le p$, then $\exists a \in S - \beta$ (i.e., $a \notin (\alpha \cap \beta)$) such that $T_1|\{x, y, a\} = T_2|\{x, y, a\} = ((x, y), a)$, and thus x and y will be in the same component after Case 2a is handled.

From the preceding discussion, it suffices to convert the trees T_1 and T_2, both defined on the leaf set $(\alpha \cap \beta)$, into millipedes T_1' and T_2', respectively. T_1' is obtained from T_1 by contracting all edges above internal nodes not in the set $\{v_1, v_2, \ldots, v_p\}$. T_2' is obtained from T_2 similarly. Thus, we have to solve the following problem now: we are given two millipedes T_1' and T_2' on the same leaf set $S' = (\alpha \cap \beta)$, where T_1' has internal nodes labeled v_1' (root of T_1') through v_p', and each v_i' has leaves corresponding to all the species in the special clusters of v_i in T_1; T_2' has internal nodes labeled u_1' (root of T_2') through u_q' and is defined similarly. Our aim is to construct a graph $G' = (V', E')$ where $V' = S'$ such that if $\exists x, y, z \in (\alpha \cap \beta)$ such that both T_1' and

T_2' resolve this triple as $((x,y),z)$ then x and y will be in the same component of G'. Once G' is known, we add the edges of G' to the edge set of G, and then the components in G will give the maximal clusters we seek.

We will show how G' can be constructed in $O(n)$ time. Consider a node v'_{i-1} in T_1' and let A be the set of leaves of v'_{i-1}. Let u'_{j-1} be the node in T_2' which is closest to u'_1 and is the parent of some species in A. Then, clearly, in G' all species in $(\alpha_{v'_i} \cap \beta_{u'_j})$ need to be in one component. For every v'_{i-1} $(2 \le i \le p)$, we will denote this intersection by the pair (v'_i, u'_j). Further, observe that if (v'_i, u'_j) and $(v'_{i'}, u'_{j'})$ are such that $v'_{i'}$ is not above v'_i and $u'_{j'}$ is not above u'_j, then $(\alpha_{v'_{i'}} \cap \beta_{u'_{j'}}) \subseteq (\alpha_{v'_i} \cap \beta_{u'_j})$.

Thus, when constructing the graph G', we need only look at all the intersections of the form (v'_i, u'_j), where for every pair of intersections $(v'_{i'}, u'_{j'})$ and (v'_i, u'_j), v'_i is closer to v'_1 than $v'_{i'}$ is, iff $u'_{j'}$ is closer to u'_1 than u'_j is.

Let $(v_1^*, u_1^*), (v_2^*, u_2^*), \ldots, (v_r^*, u_r^*)$ be the intersections we are interested in, where v_i^* is closer to v'_1 than v_{i+1}^* is $(1 \le i \le (r-1))$, and u_{j+1}^* is closer to u'_1 than u_j^* is $(1 \le j \le (r-1))$. Note that $v_1^* = v'_2$. This node and the given T_1' and T_2', uniquely determine these intersections.

In T_1', we define the *nearest parent* of a species x to be the first v_i^* to appear on the path from x to the root of T_1'. Similarly, we can define the nearest parent of a species in T_2'. The nearest parents of all the species can be computed in $O(n)$ by doing a simple traversal of T_1' and T_2'. Using the nearest parents of the species in T_1', we partition the species set into r sets $S_{v_1^*}, \ldots, S_{v_r^*}$ where $S_{v_i^*}$ contains all species which have nearest parent as v_i^*.

Observe that if any two intersections (v_i^*, u_i^*) and $(v_{i'}^*, u_{i'}^*)$ contain at least one species in common, then all the species in the two intersections need to be in the same component in G'. Inductively, if there are intersections $(v_i^*, u_i^*), \ldots, (v_j^*, u_j^*)$ such that the species in these intersections need to be in one component in G' and if there is an intersection (v_k^*, u_k^*) which has a species x in common with one of these intersections, then all the species in the intersection (v_k^*, u_k^*) need to be in the same component as the species in the intersections $(v_i^*, u_i^*), \ldots, (v_j^*, u_j^*)$. The algorithm *CONSTRUCT_G'* we present now keeps track of such an x using the variable *missing_link*, which is initialized to an $x \in (v_r^*, u_r^*)$ such that the nearest parent of x in T_2' (say u_j^*) is farthest from the root (as compared with the nearest parents of the other species in (v_r^*, u_r^*)). We will also use two additional variables: *np_missing_link* which stores u_j^* and *upper_limit* which stores v_j^*.

PROCEDURE CONSTRUCT_G'

For $i = r$ down to 1,

do{

Identify $y \in S_{v_i^*}$ such that the nearest parent of y in T_2' is
farthest away from u'_1 (i.e., root of T_2')
Let u_k^* be the nearest parent of y in T_2'; Set $z = v_k^*$
Connect all $x \in S_{v_i^*}$ to y
If *upper_limit* is not below v_i^*,
then
connect y to *missing_link*
else if (*upper_limit* is below v_i^*) or (*upper_limit* is below v_k^*)
then
set *missing_link* $= y$
np_missing_link $= u_k^*$
upper_limit $= z$

}enddo

Once we have constructed G', we can update G by setting $E(G) = E(G) \cup E(G')$. The components in G will be the maximal clusters of the RV-II. Finding the components takes $O(n)$. To recurse, we find the homeomorphic subtrees of T_1 and T_2 induced by the species in each of the maximal clusters. This can be done in $O(n)$ as previously described.

Thus the RV-II can be constructed in $O(n^2)$.

5.5. RV-III.

LEMMA 5.3. *The* RV-III *tree T of two trees T_1 and T_2 always exists and is unique. Further $C(T) = A$, where $A = \{\gamma | \gamma = \alpha \cap \beta, \alpha \in C(T_1), \beta \in C(T_2), \gamma$ compatible with $C(T_i), i = 1, 2\}$.*

Proof. We will first show that A as defined above is a compatible set. The uniqueness will then follow from the uniqueness of a set of compatible clusters [17].

Pick two clusters $\gamma_1 = \alpha_1 \cap \beta_1$ and $\gamma_2 = \alpha_2 \cap \beta_2$ such that $\gamma_i \in A; \alpha_1, \alpha_2 \in C(T_1); \beta_1, \beta_2 \in C(T_2)$. We will show that $\gamma_1 \cap \gamma_2 \in \{\emptyset, \gamma_1, \gamma_2\}$. Now, since γ_i is compatible with $C(T_1)$ and $C(T_2)$, we have $\gamma_1 \cap \alpha_2 \in \{\emptyset, \gamma_1, \alpha_2\}$. Also, we have $\gamma_1 \cap \beta_2 \in \{\emptyset, \gamma_1, \beta_2\}$. There are several cases to handle. The first case is when $\gamma_1 \subseteq \alpha_2, \gamma_1 \subseteq \beta_2$. In this case, $\gamma_1 \subseteq (\alpha_2 \cap \beta_2)$ or $\gamma_1 \cap \gamma_2 = \gamma_1$. The second case is when $\gamma_1 \supseteq \alpha_2, \gamma_1 \supseteq \beta_2$. In this case, $(\alpha_2 \cap \beta_2) \subseteq \gamma_1$ or $\gamma_1 \cap \gamma_2 = \gamma_2$. The third case is when $\gamma_1 \subseteq \alpha_2, \gamma_1 \supseteq \beta_2$. In this case, $(\alpha_2 \cap \beta_2) \subseteq \gamma_1$ and thus $\gamma_1 \cap \gamma_2 = \gamma_2$. Hence, A is a compatible set of clusters.

Now we will show that any tree T satisfying the RV-III rules will have its cluster encoding equal to A. From the third requirement for RV-III,[5] all the clusters in $C(T)$ are compatible with both $C(T_1)$ and $C(T_2)$. Now suppose we can pick a $\gamma \in C(T) - A$. This means that $\gamma \neq \alpha_i \cap \beta_j \forall \alpha_i \in C(T_1), \beta_j \in C(T_2)$. Let α_1 and β_1 be the minimal clusters in T_1 and T_2, respectively, containing γ. Clearly, $\alpha_1 \cap \beta_1 \supset \gamma$. Let u and v be the nodes in T_1 and T_2, respectively, which define the clusters α_1 and β_1. Since γ is compatible with $C(T_1)$ and $C(T_2)$, it follows that we can pick three species a, b, c such that $lca_{T_1}(a, b) = lca_{T_1}(a, c) = lca_{T_1}(b, c) = u$, $lca_{T_2}(a, b) = lca_{T_2}(a, c) = lca_{T_2}(b, c) = v$, and $a, b \in \gamma, c \in (\alpha_1 \cap \beta_1) - \gamma$. In both T_1 and T_2, the triple a, b, c is unresolved, but it is resolved as $((a, b), c)$ in T, thus contradicting the assumption that T' satifies the rules defined by RV-III. Thus we have that $C(T) \subseteq A$. Now suppose $C(T) \subset A$. Then it can be seen that we can pick a triple a, b, c which is resolved in T_1 and is either resolved the same in T_2 or is unresolved in T_2 but that a, b, c is unresolved in T. This contradicts the assumption that T satisfies the rules defined by RV-III since it does not satisfy the second (see definition of RV-III) for a maximal set of triples. Thus $C(T) = A$. ∎

LEMMA 5.4. *The RV-*III *tree T of two rooted trees can be computed in $O(n^3)$.*

Proof. We can compute $C(T)$ in $O(n^3)$ as follows. The set $X = \{\gamma | \gamma = \alpha \cap \beta, \alpha \in C(T_1), \beta \in C(T_2)\}$ can be computed in $O(n^3)$, since there are $O(n^2)$ pairs to look at and each $\alpha \cap \beta$ can be computed in $O(n)$. The set $Y = \{\gamma | \gamma \in X, \gamma$ compatible with $C(T_i)\}$ can be computed from X in $O(n^3)$, since each of the $O(n^2)$ clusters in X can be checked for compatibility with $C(T_i)$ in $O(n)$. Finally, T can be constructed from Y using the $O(n^2)$ algorithm mentioned in [17]. Thus the total time taken is $O(n^3)$. ∎

We now briefly discuss another local consensus rule that looks interesting but unfortunately does not always exist. We define $LCR1$ as a rule which requires that if

[5] If a triple a, b, c is resolved as $((a, b), c)$ in T, then it is not resolved as $(a, (b, c))$ or $((a, c), b)$ in either T_1 or T_2.

a triple a, b, c is resolved as $(a, (b, c))$ in one tree and is either resolved as $(a, (b, c))$ or unresolved in the second tree, then it is resolved as $(a, (b, c))$ in the consensus tree.

Although the above rule tries to capture the *optimistic* features of the input trees and at the same time is not a total local consensus rule, it is the case that the consensus tree defined by $LCR1$ need not exist. See Figure 7 for an example showing that $LCR2$ need not necessarily produce a tree. Figure 7(iii) shows the graph constructed by the algorithm in [3]. Since the graph is connected, it follows that the set of triple constraints does not define a tree.

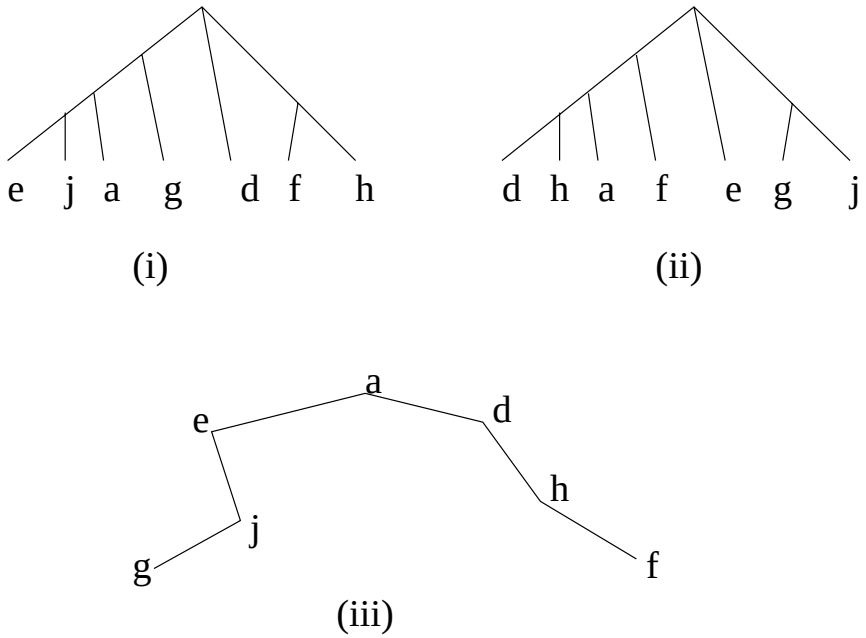


FIG. 7. *Example showing that consensus tree defined by $LCR1$ need not exist.*

6. Discussion and conclusions. Several approaches have been taken to handle the problem of resolving multiple solutions. One approach has been to find a maximum subset $S_0 \subseteq S$ inducing homeomorphic subtrees; this subtree is then called a *maximum agreement subtree* [19, 13, 24, 14]. The primary disadvantage of this approach is that it does not return an evolutionary tree on the entire species set.

The other approach which we take here requires that the resolution of the inconsistencies be represented in a single evolutionary tree for the entire species set. A classical problem in this area is the *tree compatibility problem* (also called the *cladistic character compatibility problem*) [10, 11, 12]. The tree compatibility problem says that the set $\mathcal{T}$ of trees is *compatible* if a tree T exists such that $C(T) = \cup_{T_i \in \mathcal{T}} C(T_i)$. Equivalently, if a tree T exists such that for every triple $A \subseteq S$, T resolves A iff $T|A = T_i|A$ for every $T_i \in \mathcal{T}$ which resolves A. This problem can be solved in linear time [17, 25]. The weakness of this approach is that in practice many data sets are incompatible, and it is therefore necessary to be able to handle the case where some pairs of trees resolve triples differently.

Some other approaches of this type are the *strict consensus* [4, 9] and the *median tree* [5] problems. These models are stated in terms of unrooted trees, so that instead of *clusters*, *characters* (i.e., bipartitions) on the species set are used to represent the

trees. Using the character encoding of the consensus tree as a measure of fitness to the input, the strict consensus seeks a tree with only those characters that appear in every tree in the input. The median tree, on the other hand, is defined by a metric $d(T_1, T_2)$ between rooted trees which is defined to be the cardinality of the symmetric difference of the character sets of T_1 and T_2. Given input trees $T_1, \ldots, T_k$, T is the median tree if it minimizes $\sum_i d(T, T_i)$. The median tree can be computed in polynomial time and has a nice characterization in terms of the character encoding [5, 23, 9]. Both the above notions are related to versions of the local consensus problem (for example, the relaxed versions RV-I and RV-III), and the relevant local consensus trees in many cases contain at least as much "information" as these trees.

The work represented in this paper can be extended in several directions. As we have noted, for all local consensus functions the local consensus tree of a set of k trees can be computed in time polynomial in k and $n = |S|$. Many of these local consensus trees can be constructed in $O(kn)$ time.

REFERENCES

[1] E. Adams III, *N-trees as nestings: Complexity, similarity, and consensus*, J. Classification, 3 (1986), pp. 299–317.

[2] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison–Wesley, Reading, MA, 1974.

[3] A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman, *Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions*, SIAM J. Comput., 10 (1981), pp. 405–421.

[4] J. Barthélemy and F. Janowitz, *A formal theory of consensus*, SIAM J. Discrete Math., 3 (1991), pp. 305–322.

[5] J. Barthélemy and F. McMorris, *The median procedure for n-Trees*, J. Classification, 3 (1986), pp. 329–334.

[6] W. Brown, E. M. Prager, A. Wang, and A. C. Wilson, *Mitochondrial DNA sequences of primates: Tempo and mode of evolution*, J. Mol. Evol., 18 (1982), pp. 225–239.

[7] D. Bryant and M. Steel, *Extension operations on sets of leaf-labelled trees*, Research report 118, Department of Mathematics and Statistics, University of Canterbury, Christchurch, New Zealand, 1994.

[8] H. Colonius and H. H. Schulze, *Tree structures for proximity data*, British J. Math. Statist. Psych., 34 (1981), pp. 167–180.

[9] W. H. E. Day, *Optimal algorithms for comparing trees with labeled leaves*, J. Classification, 2 (1985), pp. 7–28.

[10] G. F. Estabrook, C. S. Johnson, Jr., and F. R. McMorris, *An idealized concept of the true cladistic character*, Math. Biosci., 23 (1975), pp. 263–272.

[11] G. F. Estabrook, C. S. Johnson, Jr., and F. R. McMorris, *An algebraic analysis of cladistic characters*, Discrete Math., 16 (1976), pp. 141–147.

[12] G. F. Estabrook, C. S. Johnson, Jr., and F. R. McMorris, *A mathematical foundation for the analysis of cladistic character compatibility*, Math. Biosci., 29 (1976), pp. 181–187.

[13] M. Farach and M. Thorup, *Optimal evolutionary tree comparison by sparse dynamic programming*, in Proc. 35th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Piscataway, NJ, November 1994, pp. 770–779.

[14] M. Farach, T. Przytycka, and M. Thorup, *On the agreement of many trees*, Inform. Process. Lett., 55 (1995), pp. 297–301.

[15] J. Felsenstein, *Numerical methods for inferring evolutionary trees*, Quart. Review of Biology, 57 (1982), pp. 379–404.

[16] C. R. Finden and A. D. Gordon, *Obtaining common pruned trees*, J. Classification, 2 (1985), pp. 225–276.

[17] D. Gusfield, *Efficient algorithms for inferring evolutionary trees*, Networks, 21 (1991), pp. 19–28.

[18] S. Kannan, E. Lawler, and T. Warnow, *Determining the evolutionary tree using experiments*, J. Algorithms, 21 (1996), pp. 26–50.

[19] D. Keselman and A. Amir, *Maximum agreement subtree in a set of evolutionary trees—Metrics and efficient algorithms*, in Proc. 35th Annual Symposium on Foundations of Com-

puter Science, IEEE Computer Society Press, Piscataway, NJ, November 1996, pp. 758–769.
[20] D. Harel and R. Tarjan, *Fast algorithm for finding nearest common ancestors*, SIAM J. Comput., 13 (1984), pp. 338–355.
[21] M. Henzinger, V. King, and T. Warnow, *Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology*, in Proc. 7th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM/SIAM, January 28–30, 1996, pp. 333–340.
[22] G. Nelson, *Cladistic analysis and synthesis: Principles and definitions, with a historical note on Adanson's Famille des Plantes* (1763–1764), Systematic Zoology, 28 (1979), pp. 1–21.
[23] F. McMorris and M. Steel, *The complexity of the median procedure for binary trees*, in Proc. 4th Conference of the International Federation of Classification Societies, Paris, 1993; Stud. Classification Data Anal. Knowledge Organ., by Springer-Verlag, to appear.
[24] M. Steel and T. Warnow, *Kaikoura tree theorems: Computing the maximum agreement subtree*, Inform. Process. Lett., 48 (1993), pp. 77–82.
[25] T. Warnow, *Tree compatibility and inferring evolutionary history*, J. Algorithms, 16 (1994), pp. 388–407.

SIAM J. COMPUT.
Vol. 27, No. 6, pp. 1725–1746, December 1998
© 1998 Society for Industrial and Applied Mathematics
010

PARALLEL ALGORITHMS WITH OPTIMAL SPEEDUP FOR BOUNDED TREEWIDTH*

HANS L. BODLAENDER† AND TORBEN HAGERUP‡

Abstract. We describe the first parallel algorithm with optimal speedup for constructing minimum-width tree decompositions of graphs of bounded treewidth. On n-vertex input graphs, the algorithm works in $O((\log n)^2)$ time using $O(n)$ operations on the EREW PRAM. We also give faster parallel algorithms with optimal speedup for the problem of deciding whether the treewidth of an input graph is bounded by a given constant and for a variety of problems on graphs of bounded treewidth, including all decision problems expressible in monadic second-order logic. On n-vertex input graphs, the algorithms use $O(n)$ operations together with $O(\log n \log^* n)$ time on the EREW PRAM, or $O(\log n)$ time on the CRCW PRAM.

Key words. graph algorithms, parallel algorithms, treewidth, tree decomposition, monadic second-order logic, graph reduction, derandomization, pathwidth, partial k-trees

AMS subject classifications. 68Q20, 68Q22, 68Q25, 68R10, 05C05, 05C85

PII. S0097539795289859

1. Introduction.

The concept of treewidth has proved to be a useful tool in the design of graph algorithms: many important classes of graphs have bounded treewidth, and many important graph problems that are otherwise quite hard can be solved efficiently on graphs of bounded treewidth. A *tree decomposition* of an undirected graph $G = (V, E)$ is a pair $(T, \mathcal{U})$, where $T = (X, F)$ is a tree and $\mathcal{U} = \{U_x \mid x \in X\}$ is a family of subsets of V called *bags*, one for each node in T, such that

(i) $\bigcup_{x \in X} U_x = V$ (every vertex in G occurs in some bag);

(ii) for all $(v, w) \in E$, there exists an $x \in X$ such that $\{v, w\} \subseteq U_x$ (every edge in G is "internal" to some bag);

(iii) for all $x, y, z \in X$, if y is on the path from x to z in T, then $U_x \cap U_z \subseteq U_y$ (every vertex in G occurs in the bags in a connected part of T, i.e., in a subtree).

The *width* of a tree decomposition $(T, \{U_x \mid x \in X\})$ is $\max_{x \in X} |U_x| - 1$. The *treewidth* of a graph G, denoted $tw(G)$, is the smallest treewidth of any tree decomposition of G. *Path decompositions* and *pathwidth* are defined analogously, with the tree T restricted to be a path.

As mentioned above, many well-known classes of graphs are of bounded treewidth; i.e., for each class there is a uniform upper bound on the treewidth of all graphs in the class. Such classes include those of trees, forests, outerplanar graphs, k-outerplanar graphs (for every fixed $k \geq 1$), Halin graphs, series-parallel graphs, graphs that avoid a planar graph H as a minor (for every fixed H), and graphs of bounded bandwidth. For an overview, see [11].

*Received by the editors August 2, 1995; accepted for publication (in revised form) September 20, 1996; published electronically June 3, 1998. This research was partially supported by the ESPRIT Basic Research Actions Program of the EU under contract 7141 (project ALCOM II). A preliminary version of this paper was presented at the 22nd International Colloquium on Automata, Languages and Programming (ICALP) in July 1995.

http://www.siam.org/journals/sicomp/27-6/28985.html

†Department of Computer Science, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, the Netherlands (hansb@cs.ruu.nl).

‡Max-Planck-Institut für Informatik, Im Stadtwald, D–66123 Saarbrücken, Germany (torben@mpi-sb.mpg.de).

The majority of efficient algorithms for graphs of bounded treewidth depend not only on a guarantee that the treewidth of an input graph is small, but in fact on the availability of a minimum-width tree decomposition of the input graph, so that the construction of minimum-width tree decompositions for graphs of bounded treewidth is a key problem. A quest for the fastest possible algorithm for this problem [5, 43, 22, 10, 38, 35, 42, 8] led to the linear-time algorithm of [8], which eliminated the bottleneck in a large number of algorithms for bounded treewidth.

In the setting of parallel computation, the situation is similar. Many otherwise difficult graph problems can be solved in NC (i.e., in polylogarithmic time with a polynomial amount of hardware) on graphs of bounded treewidth, and again the need for a minimum-width tree decomposition is a serious bottleneck. The best parallel algorithms for computing tree decompositions of width k of graphs of treewidth k, for fixed k, run on the CRCW PRAM using $O((\log n)^2)$ time and $O(n^{2k+5})$ processors [15, 16] or $O(\log n)$ time and $O(n^{3k+4})$ processors [7]. Although these algorithms are fast, they are extremely wasteful in terms of processors, in view of the linear-time sequential algorithm. For the cases $k = 2$ and $k = 3$, somewhat more efficient algorithms using $O((\log n)^2)$ time and $O(n^{k+1})$ processors were described by Granot and Skorin-Kapov [27]. A related result was obtained by Wanke [45], who showed that the problem of deciding whether the treewidth of an input graph is bounded by a constant k belongs to the complexity class LOGCFL; this result also does not seem to lead to parallel algorithms that are efficient from the point of view of processor utilization. If we relax the requirements by allowing tree decompositions of width $O(k)$, rather than exactly k, more algorithms come into play: Lagergren [35] finds a decomposition of width $\leq 8k+7$ in $O((\log n)^3)$ time using $O(n)$ processors, and we believe that Reed's sequential $O(n \log n)$-time algorithm [42] for obtaining a decomposition of width $\leq 4k + 3$ can be parallelized (using an algorithm of Khuller and Schieber [33] to solve a central path-finding problem) to yield an algorithm that works in $O((\log n)^2)$ time using $O(n\alpha(n)/\log n)$ processors, where α is a very slowly growing "inverse Ackermann" function. The parallel version of Reed's algorithm uses $O(n\alpha(n) \log n)$ *operations*, i.e., has a time-processor product of $O(n\alpha(n) \log n)$, and is the most efficient of the parallel algorithms discussed above. Still, since the problem can be solved in linear sequential time, the algorithm does not have optimal speedup, which requires a time-processor product of $O(n)$.

We describe an EREW PRAM algorithm for constructing minimum-width tree decompositions for graphs of bounded treewidth in $O((\log n)^2)$ time using $O(n)$ operations. The algorithm achieves optimal speedup and is the first parallel algorithm to do so. Moreover, the new algorithm is second in speed only to Bodlaender's algorithm [7] but uses a weaker model of computation (the EREW PRAM versus the CRCW PRAM), on which Bodlaender's algorithm can be simulated only in the same time of $O((\log n)^2)$. The new result immediately implies that a large number of problems on graphs of bounded treewidth can now be solved by parallel algorithms with optimal speedup.

A subroutine used in the construction algorithm but of independent interest is a parallel version of an algorithm due to Bodlaender and Kloks [13]. The algorithm takes as input a tree decomposition of bounded width of a graph G and outputs a minimum-width tree decomposition of G, thus blurring the distinction between the "exact" and the "approximate" construction algorithms discussed above. The new algorithm runs in $O(\log n)$ time using $O(n)$ operations on the EREW PRAM.

While we cannot compute tree decompositions faster than in $O((\log n)^2)$ time, it turns out that we can give faster algorithms for the related problem of deciding whether the treewidth of an input graph is bounded by a given constant k. The algorithms have optimal speedup (i.e., use $O(n)$ operations) and run in $O(\log n \log^* n)$ time on the EREW PRAM, or in $O(\log n)$ time on the CRCW PRAM. We achieve the same resource bounds for a number of problems on graphs of bounded treewidth, including all problems expressible in monadic second-order logic. These algorithms operate without an explicit tree decomposition and so bypass the (time) bottleneck of our construction algorithm. Furthermore, we achieve the same results for path decompositions and pathwidth as for tree decompositions and treewidth.

The paper combines several different techniques of wide applicability. The *graph-reduction* technique consists in repeatedly replacing parts of the graph at hand by simpler parts until a trivial graph results. The problem of interest is then solved for the trivial graph, and the solution is "carried along" while the changes to the graph are undone in reverse order. This technique pervades the paper and is used in the construction algorithm as well as in the decision algorithms (and also in the width-minimizing algorithm, provided that tree contraction is viewed as a special case of graph reduction).

Another technique used in the derivation of the CRCW PRAM decision algorithm from the corresponding EREW PRAM algorithm is that of *derandomization*. The basic idea of derandomization is that instead of letting random coin tosses select one algorithm to be executed from a collection of deterministic algorithms, we execute all deterministic algorithms in the collection and pick the best output. Because of the need to simulate several possible executions, derandomization is often associated with a price in the form of increased resource requirements. Here we use derandomization to derive a parallel algorithm with optimal speedup; i.e., we pay no price.

A third technique of less general applicability but nonetheless independent interest is that of *bounded adjacency-list search*, which tries to circumvent the difficulties caused by high-degree vertices in parallel algorithms for sparse graphs by letting each neighbor of a high-degree vertex v inspect only a piece of constant size of the adjacency list of v near its own entry, rather than the whole adjacency list. The bounded adjacency-list search technique was used previously (although not named) in [28]; there, as here, the technique serves to eliminate the need for both concurrent reading and writing and for superlinear space.

Unlike certain related results, most of our algorithms are explicit and do not rely on nonconstructive arguments. Only the results of Theorems 5.1 and 6.1 are nonconstructive, but can be made constructive in many concrete cases, as discussed near the end of section 5. On the other hand, it should be noted that large constant factors prevent our algorithms from being practical.

All graphs in this paper are undirected, loopless, and without multiple edges. We assume that all graphs, not excluding trees, are represented according to an *adjacency-list* representation: each vertex v in an n-vertex graph G is represented by an integer name of size $O(n)$ and has a pointer to a doubly linked adjacency list with an entry for each neighbor of v in G. For each neighbor w of v, the entry of w in v's adjacency list contains the name of w as well as a *cross pointer* to the entry of v in w's adjacency list.

2. Minimizing decomposition width. In this section we show how to obtain a minimum-width tree decomposition of a graph G from any tree decomposition of G of bounded width. We begin with an observation that allows us to assume that

tree decompositions are rooted, binary, and of logarithmic depth whenever this is convenient. In representation terms, every nonroot node in a rooted tree knows its parent, and a rooted tree is *binary* if no node has more than two children.

We appeal twice to the *tree-contraction* technique introduced by Miller and Reif [39], which we therefore describe in generic terms. Applied to an n-node binary input tree $T = (V, E)$, a tree-contraction algorithm produces a sequence $T = T_0 = (V_0, E_0), T_1 = (V_1, E_1), \ldots, T_r = (V_r, E_r)$ of $O(\log n)$ trees, ending with a one-node tree T_r, such that each tree T_i, for $i = 1, \ldots, r$, is obtained from its predecessor T_{i-1} in the sequence by contracting a set of edges $F_{i-1} \subseteq E_{i-1}$ with the following properties:

(i) F_{i-1} spans a matching (i.e., no node in V_{i-1} is incident to more than one edge in F_{i-1});

(ii) each edge in F_{i-1} has at least one endpoint with at most one child in T_{i-1}.

A sequence $T_0, \ldots, T_r$ with these properties, called a *contraction sequence* for T, can be computed in $O(\log n)$ time on an EREW PRAM using $O(n)$ operations and $O(n)$ space [1, 18, 24, 25, 34] (the connection to our generic description of tree contraction is easiest to establish in the case of the simple and elegant algorithm of [1, 34]). The second property mentioned above implies that each tree in the sequence is binary.

Let $X = \bigcup_{i=0}^{r} V_i$. We can define a rooted, binary tree T' on the node set X, called the *contraction tree* corresponding to the sequence $T_0, \ldots, T_r$, in the following way: the nodes in V, which will be called *base nodes*, are the leaves of T', and whenever a node $x \in X$ results from the contraction of an edge (y, z), we make x the parent of y and z in T'. For all $x, y \in X$, we will say that x *contains* y if x is an ancestor of y in T'. The base nodes contained in any node in X span a connected subgraph of the input tree T. For $i = 1, \ldots, r$, we root T_i at the node in V_i containing the root of T.

For $i = 0, \ldots, r$, call two base nodes v and w *i-neighbors* if $(v, w) \in E$ and v and w are not contained in the same node in V_i. For each $x \in V_i$, the i-neighbors of base nodes contained in x are contained in distinct neighbors of x in T_i. For all $x \in X$, we define the *border* of x as the set of base nodes contained in x and adjacent in T to one or more base nodes not contained in x. For $i = 0, \ldots, r$, if $x \in V_i$, then a base node contained in x belongs to the border of x precisely if it has at least one i-neighbor; in particular, the border of x contains at most three nodes.

Lemmas 2.1 and 2.2 below slightly improve a result of [7] by employing a more efficient subroutine; the same improvement was observed in [31].

LEMMA 2.1. *The following problem can be solved on an EREW PRAM using $O(\log n)$ time, $O(n)$ operations, and $O(n)$ space: given an n-node rooted, binary tree T, compute a rooted, binary tree decomposition of T of depth $O(\log n)$ and width at most* 2.

Proof. Begin by using tree contraction to compute a contraction sequence $T = T_0 = (V_0, E_0), T_1 = (V_1, E_1), \ldots, T_r = (V_r, E_r)$ for the input tree $T = (V, E)$ and construct the corresponding contraction tree $T' = (X, F)$.

Observe that if e and e' are the edges incident on a node v of degree 2 in some tree H, then the tree obtained from H by contracting e is the same as the tree obtained from H by contracting e' (the contraction can be "flipped" to the other side of v). Because of this, we can assume without loss of generality that when an edge between a node v of degree 2 and a node w of degree 3 is contracted in the transition from T_{i-1} to T_i, for some i with $1 \leq i \leq r$, then v is the parent of w in T_{i-1}. To see this, first note that all edge contractions that violate the assumption—we will call these $(3, 2)$-*contractions*—can be carried out separately (i.e., we replace the one-step transition from T_{i-1} to T_i by a two-step process, thereby doubling r). Now each $(3, 2)$-

contraction can be "flipped," as described above, without changing the resulting tree; note that the edge set of the "flipped" contractions still spans a matching.

For all $x \in X$, denote the border of x by $B(x)$. We next show that no border contains three distinct vertices.

CLAIM 1. *For all $x \in X$, $|B(x)| \leq 2$.*

Proof. Suppose that for some i with $1 \leq i \leq r$, some base node $v \in V$ has two i-neighbors and belongs to $B(x)$ for some $x \in V_i$ with $|B(x)| \geq 2$. Then neither of the two i-neighbors of v is its parent in T. To see why this is true, let j be minimal such that v belongs to $B(y)$ for some $y \in V_j$ with $|B(y)| \geq 2$. It can be seen that then y must, in fact, result from the contraction of an edge (v, z), where $z \in V_{j-1}$ is of degree 2 in T_{j-1}. But then, by the absence of $(3, 2)$-contractions, z must be the parent of v in T_{j-1}, which implies that the parent of v in T is not an i-neighbor of v for any $i \geq j$.

Since the claim $|B(x)| \leq 2$ is trivially true for all base nodes x, assume by induction that it is true for all $x \in V_{i-1}$, for some i with $1 \leq i \leq r$, and consider a node $x \in V_i$ resulting from the contraction of an edge (y, z), where $y, z \in V_{i-1}$. Since $|B(x)|$ is bounded by the degree of x in T_i, which is two less than the sum of the degrees of y and z in T_{i-1}, we can assume that y is of degree 3 and that z is of degree 2 in T_{i-1}. We will show that only one node in each of $B(y)$ and $B(z)$ also belongs to $B(x)$, from which $|B(x)| \leq 2$ follows immediately. In the case of $B(z)$, this is easy to see: the nodes in $B(z)$ have a total of two $(i-1)$-neighbors, and one of these is not an i-neighbor. Similarly, the nodes in $B(y)$ lose one of their three $(i-1)$-neighbors. We must show that the two remaining $(i-1)$-neighbors, which are also i-neighbors, are adjacent to the same node in $B(y)$. But if this is not the case, then $|B(y)| = 2$ and $B(y)$ contains a node v with two $(i-1)$-neighbors, one of which belongs to $B(z)$. By what was shown above, neither of the $(i-1)$-neighbors of v is its parent in T. But this contradicts the fact that z must be the parent of y in T_{i-1} by the absence of $(3, 2)$-contractions. This ends the proof of the claim. ∎

The sets $B(x)$, where $x \in X$, can be computed as follows: successively, for $i = 0, \ldots, r$, we compute $B(x)$, along with the set of i-neighbors of all nodes in $B(x)$, for all $x \in V_i$. This is trivial for $i = 0$, and if, for some i with $1 \leq i \leq r$, a node $x \in V_i$ is obtained by contracting an edge (y, z), where $y, z \in V_{i-1}$, then $B(x) \subseteq B(y) \cup B(z)$, and an $(i-1)$-neighbor of a base node $v \in B(y) \cup B(z)$ is also an i-neighbor of v exactly if it does not belong to $B(y) \cup B(z)$, so that the information pertaining to x can easily be derived from the information pertaining to y and z.

We now associate a set $U_x \subseteq V$ with each $x \in X$ as follows: if x is a base node, i.e., a leaf in T', we take $U_x = \{x\}$. Otherwise, if x has the children y and z in T', we take $U_x = B(y) \cup B(z)$. We will show that $(T', \{U_x \mid x \in X\})$ is a tree decomposition of T. First, because of the convention regarding leaves of T', the condition $\bigcup_{x \in X} U_x = V$ is trivially satisfied. Second, for every edge (v, w) in E, it is easy to see that $\{v, w\} \subseteq U_x$, where x is the lowest common ancestor of v and w in T'. And, third, the set of nodes whose bags contain a base node v form an initial part of the path in T' from v to the root of T', and hence span a connected subgraph of T'.

The width of the tree decomposition defined above is bounded by one less than twice the maximum size of a border. By the claim shown above, this means that the width is at most 3. We now describe how to reduce the width to at most 2. Suppose that $x \in X$ is a node in T' whose associated bag U_x is of size 4, let y and z be the children of x in T', and take $B(y) = \{v_1, v_2\}$ and $B(z) = \{v_3, v_4\}$, so that $U_x = \{v_1, v_2, v_3, v_4\}$. As follows easily from arguments used to bound the sizes of

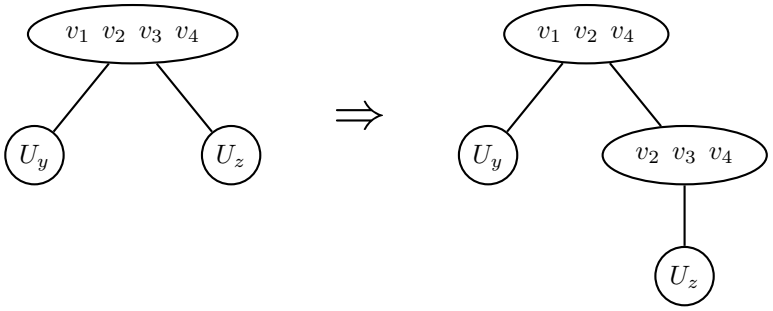


FIG. 1. *Transforming from width* 3 *to width* 2.

all bags by 2, $B(x)$ "inherits" exactly one element of each of $B(y)$ and $B(z)$, so that we can assume that $B(x) = \{v_1, v_4\}$. Then v_2 and v_3 do not occur in the bag of the parent of x in T', if any. Moreover, $(v_2, v_3) \in E$ and hence $(v_1, v_3) \notin E$. It is now easy to see that the transformation illustrated in Figure 1 preserves the defining properties of a tree decomposition. Applied at all nodes with bags of size 4, it produces a new tree decomposition of T of width at most 2. The depth increases by a factor of at most 2 and hence remains $O(\log n)$, as desired.

Starting from the sequence $T_0, \ldots, T_r$, the algorithm constructs the tree T', then computes the sets $B(x)$ and U_x for all $x \in X$, and finally carries out the transformation of T' described above. Each of these steps can easily be done in $O(\log n)$ time using $O(n)$ operations and $O(n)$ space. □

In the lemma below as well as in several later results, the input parameter k is qualified as being a constant, the meaning of which is that k can be any positive integer but that the $O(\cdots)$ of the result may (and will) hide factors that depend on k.

LEMMA 2.2. *For all constants $k \geq 1$ and all integers $n \geq 2$, the following problem can be solved on an EREW PRAM using $O(\log n)$ time, $O(n)$ operations, and $O(n)$ space*: *given a tree decomposition with n nodes and of width k of a graph G, compute a rooted, binary tree decomposition of G of depth $O(\log n)$ and width at most $3k + 2$.*

Proof. We begin by replacing each node of degree $m \geq 4$ and with associated bag U in the given tree decomposition by a path of $m - 2$ nodes, each of degree 3 and with associated bag U, which obviously preserves the defining properties of a tree decomposition. Then we construct an Euler tour of the modified tree (see [44]) and root the tree by breaking the Euler tour at an arbitrary node of degree at most 2, declared to be the root, computing the distance from each node to the root along the Euler tour by means of list ranking [17, 4], and determining the parent of each nonroot node as the neighbor with a smaller distance to the root. After these preliminary steps, which can easily be carried out within the stated resource bounds, we can assume that the input is a tree decomposition $(T, \mathcal{U})$, where $T = (V, E)$ is rooted and binary.

Now use the algorithm of Lemma 2.1 to obtain a rooted, binary tree decomposition $(T', \mathcal{Q})$ of T of width at most 2 and depth $O(\log n)$. Replacing each node in a bag in $\mathcal{Q}$ by the vertices of G in its own associated bag, we obtain the desired tree decomposition of G. More precisely, write $\mathcal{U} = \{U_v \mid v \in V\}$, $T' = (X, F)$, and $\mathcal{Q} = \{Q_x \mid x \in X\}$ and take $R_x = \bigcup_{v \in Q_x} U_v$ for all $x \in X$. Then $(T', \{R_x \mid x \in X\})$ is a tree decomposition of G. For if a vertex u of G occurs in U_v, with $v \in V$, and $v \in Q_x$, with $x \in X$, then $u \in R_x$. Similarly, both endpoints of each edge in G occur in some bag U_v, with $v \in V$, and therefore also in some bag R_x, with $x \in X$. Finally, each

vertex u of G occurs in the bags U_v in a subtree of T, and two adjacent nodes in this subtree occur in the bags Q_x in overlapping subtrees of T', so that, altogether, u occurs in the bags R_x in a connected subgraph of T'. The width of the tree decomposition $(T', \{R_x \mid x \in X\})$ is at most $(2+1)(k+1)-1 = 3k+2$. □

We will use the phrase "balancing a tree decomposition" to describe an application of the algorithm implicit in the preceding lemma. The remaining goal in the present section is to prove the result below.

THEOREM 2.3. *For all constants $k \geq 1$ and all integers $n \geq 2$, the following problem can be solved on an EREW PRAM using $O(\log n)$ time, $O(n)$ operations, and $O(n)$ space: given a tree decomposition with n nodes and of width k of a graph G, construct a minimum-width tree decomposition of G.*

The corresponding decision problem (is $tw(G) \leq l$?, for some given l) can be solved by a straightforward parallelization of the decision algorithm of [13]. The latter algorithm consists of a pass from the leaves to the root of a tree decomposition of the input graph, which, in light of Lemma 2.2, can be taken to be binary and of logarithmic depth. The processing of each node takes constant time, and all nodes on the same level in the tree can be processed in parallel. If the nodes in the tree decomposition are first sorted by their levels, which can be done in $O(\log n)$ time using $O(n)$ operations [41, Lemma 3.1], it is easy to process the whole tree in $O(\log n)$ time using $O(n)$ operations. The sequential construction (as opposed to decision) algorithm processes the tree decomposition in three passes. In one of these, the processing of a node no longer necessarily takes constant time, so that an amortization argument is used in [13] to bound the total running time by $O(n)$. Since this appears to stand in the way of a direct parallelization, we choose a somewhat different approach.

Suppose that the input graph is $G = (V, E)$. Close inspection of the algorithm of [13] (we omit the details, some of which were hinted at above) reveals that $O(\log n)$ time and $O(n)$ operations suffice to compute a certain *implicit representation* of the desired tree decomposition $(T = (X, F), \{U_x \mid x \in X\})$ consisting of the binary tree T (without the bags U_x) together with, for each $v \in V$, a collection $\mathcal{P}_v$ of disjoint simple paths in T whose union contains a node $x \in X$ if and only if $v \in U_x$. Rather than directly specifying the set of vertices contained in each bag, the implicit representation thus presents the set of bags containing each vertex in the form of a collection of disjoint paths. For each $v \in V$, a path in $\mathcal{P}_v$ with endpoints x and y is represented by marking both x and y with the triple (x, y, v); a node may be marked with several triples but, of course, with at most $k+1$ triples. We clearly have $|X| = O(n)$, since the size of the tree T cannot exceed the number of operations used to compute it.

By the preceding discussion, proving Theorem 2.3 boils down to showing the following lemma.

LEMMA 2.4. *For all constants $k \geq 1$ and all integers $n \geq 2$, the following problem can be solved on an EREW PRAM using $O(\log n)$ time, $O(n)$ operations, and $O(n)$ space: given a rooted, binary n-node tree T and a collection $\mathcal{P}$ of simple paths in T, each of which is labeled by an integer and represented, at each of its endpoints, by a triple specifying its endpoints and label, such that no node in T belongs to more than $k+1$ paths in $\mathcal{P}$, mark each node x in T with the set of all labels of paths in $\mathcal{P}$ containing x.*

Proof. Since duplicates are easily eliminated, we can assume that no two paths in $\mathcal{P}$ have both the same endpoints and the same label, so that we can identify each path with the triple marking its endpoints. We will also assume that the endpoints of each path are distinct, since paths consisting of a single node are trivial to handle.

We begin by using tree contraction to obtain a contraction sequence $T = T_0 = (V_0, E_0), \ldots, T_r = (V_r, E_r)$ for the input tree $T = (V, E)$. We will process the sequence twice, first in the order of increasing indices (the *up phase*), and then in the order of decreasing indices (the *down phase*).

Let $X = \bigcup_{i=0}^{r} V_i$. During the up phase, we associate a set $S(x)$ with each node $x \in X$. For $x \in V$, $S(x)$ is the set of paths in $\mathcal{P}$ with x as an endpoint. For $i = 1, \ldots, r$, if a node $x \in V_i$ results from the contraction of an edge (v, w), where $v, w \in V_{i-1}$, we compute $S(x)$ as $(S(v) \cup S(w)) \backslash (S(v) \cap S(w))$; it is easy to see by induction that for all $x \in X$, $S(x)$ will be the set of paths in $\mathcal{P}$ with exactly one endpoint contained in x. During the down phase, for $i = r, \ldots, 1$, we modify $S(v)$ for all $v \in V_{i-1} \backslash V_i$ as follows: suppose that v and another node $w \in V_{i-1}$ are both contained in $x \in V_i$. Then, for each pair (y, z) of neighbors of x in T_i such that some node in V_{i-1} contained in y is separated in T_{i-1} from some node in V_{i-1} contained in z by the removal of v (i.e., v is "between" y and z), add to $S(v)$ all paths in $S(y) \cap S(z)$. Again, it is not difficult to see by backward induction on i that the final value of $S(x)$, for all $x \in X$, will be the set of paths in $\mathcal{P}$ comprising at least one node contained in x and at least one node not contained in x. In particular, the values $S(v)$, where $v \in V$, precisely constitute the desired output. Since the number of paths containing a given node in V is bounded by a constant, and since each path in $S(x)$ must contain at least one of the at most three border nodes of x, for all $x \in X$, each set $S(x)$ is of constant size, and the whole computation can be carried out in $O(\log n)$ time using $O(n)$ operations and $O(n)$ space. □

With a similar (but, in fact, easier) argument one can also show the result below.

THEOREM 2.5. *For all constants $k, l \geq 1$ and all integers $n \geq 2$, the following problem can be solved on an EREW PRAM using $O(\log n)$ time, $O(n)$ operations, and $O(n)$ space*: *given a tree decomposition with n nodes and of width k of a graph G, decide whether the pathwidth of G is at most l and, if so, construct a minimum-width path decomposition of G.*

3. A structural lemma. In this section we provide the basis for showing that any sufficiently large connected graph of bounded treewidth admits a large number of reductions of certain types. Moreover, given any adjacency-list representation of the graph, a large fraction of these reductions can be identified efficiently.

A well-known fact that we shall use below is that every n-vertex graph of treewidth $\leq k$ contains at most kn edges for all positive integers n and k. We provide a brief proof. Since removing a vertex from a graph of treewidth $\leq k$ with at least two vertices leaves a graph of treewidth $\leq k$, it suffices to show that every graph G of treewidth $\leq k$ contains a vertex of degree $\leq k$. To this end consider a tree decomposition $(T = (X, F), \{U_x \mid x \in X\})$ of G of width $\leq k$ with a minimal number of nodes (i.e., $|X|$ is minimal over all such tree decompositions) and pick a node $x \in X$ of degree ≤ 1 in T. U_x contains at least one vertex v that does not occur in any other bag (otherwise x would be superfluous), and v has at most k neighbors, as desired (they all belong to U_x).

The *boundary* of a subgraph H of a graph G is the set of those vertices in G that have at least one neighbor in H but do not themselves belong to H. Let d, k, $n_{\min}$, and $n_{\max}$ be positive integers to be characterized more closely in the following. A vertex will be called *small* if its degree is bounded by d, and *large* otherwise. Given a graph G of treewidth at most k, we are essentially looking for connected subgraphs of G consisting of between $n_{\min}$ and $n_{\max}$ small vertices and with a boundary of size at most $2(k+1)$. It turns out that such subgraphs may not occur in G at all, for which

reason we have to replace the connectedness condition by a weaker, more complicated condition described below after the introduction of additional terminology.

Two vertices are said to be *twins* if they have the same set of neighbors. By analogy, two subgraphs of a common graph are called twins if they have the same boundary. A *weakly connected component* of a subgraph H of a graph G is a connected component of the graph obtained from H by the introduction of an edge between each pair of nonadjacent vertices in H with a common small neighbor in G; a weakly connected component of H may thus comprise several (usual) connected components of H, linked indirectly via small common neighbors in the boundary of H. A subgraph that consists of a single weakly connected component is *weakly connected*. Given an adjacency-list representation of a graph G, two disjoint subgraphs H_1 and H_2 of G are said to be *acquainted* if the intersection of their boundaries contains a vertex in whose adjacency list some entry of a vertex in H_1 is separated from some entry of a vertex in H_2 by a distance of at most d. This definition, which embodies the bounded adjacency-list search technique, reflects the fact that H_1 can "discover" H_2 by searching through a piece of length at most $2d + 1$ of the adjacency list of each of its boundary vertices.

We can now define the objects of interest and state the main result of the section. A *valley* in a graph G is a weakly connected subgraph of G induced by a set of at most $n_{\max}$ small vertices and with a boundary of size at most $2(k + 1)$. A *plain* (or $(d, k, n_{\min}, n_{\max})$-plain, for emphasis) in G (relative to a particular adjacency-list representation of G) is a subgraph of G, induced by a set of at least $n_{\min}$ and at most $n_{\max}$ vertices, whose weakly connected components are pairwise acquainted twin valleys.

LEMMA 3.1. *For all integer constants $k, n_{min} \geq 1$, there are constants $d, n_{max} \geq 1$ and $c > 0$ such that every connected graph with $n > n_{max}$ vertices and treewidth at most k contains at least cn disjoint (d, k, n_{min}, n_{max})-plains (relative to any adjacency-list representation).*

Proof. Take $b = 3(k + 1)(n_{\min} + 1)$. We will prove the lemma with $n_{\max} = 3b$ and $d = 2^{k+4} n_{\min} n_{\max}$. Let $G = (V, E)$ be a connected graph with $n > n_{\max}$ vertices and treewidth at most k and fix a particular adjacency-list representation of G and a particular maximal collection $\mathcal{P}$ of disjoint $(d, k, n_{\min}, n_{\max})$-plains in G (the complexity of computing such a collection is not an issue here, since we are concerned with an assertion of mere existence). We will show that $|\mathcal{P}| \geq cn$ for a suitably chosen constant $c > 0$.

Let $(T, \mathcal{U})$ be a tree decomposition of G of width at most k and write $T = (X, F)$ and $\mathcal{U} = \{U_x \mid x \in X\}$. We view T as rooted at an arbitrary node. Using the same standard transformation as in the beginning of the proof of Lemma 2.2, we can assume without loss of generality that T is binary. On two occasions in the proof we will use the fact that if $v \in V$, then the subgraph T_v of T induced by the node set $\{x \in X \mid v \in U_x\}$ is a tree, whose root can therefore be reached from any node in T_v by going from a child node to a parent node zero or more times; we will refer to this as the *root-seeking principle*.

We begin by showing that the set X of tree nodes can be partitioned into disjoint *clusters* $C_1, \ldots, C_s$ such that for $i = 1, \ldots, s$ the following conditions hold:

1. C_i induces a subtree of T;
2. $|\bigcup_{x \in C_i} U_x| \leq n_{\max}$;
3. if C_i does not contain the root of T, then $|\bigcup_{x \in C_i} U_x| \geq b$.

The partition $C_1, \ldots, C_s$ can be constructed by a simple procedure that processes T

in inverse topological order; i.e., every node is processed after all of its children. The processing of a node y computes the set C consisting of y itself and all descendants of y that have not yet been assigned to clusters. If $|\bigcup_{x\in C} U_x| \geq b$ or y is the root of T, then C is made into a new cluster; otherwise the processing continues to the next node.

It is easy to see that the set of nodes assigned to a cluster always induces a connected subgraph of T; i.e., condition 1 above is satisfied. Condition 3 is satisfied by construction. As for condition 2, observe that if a cluster C is formed during the processing at a node y, then C receives a contribution of at most $b-1$ vertices from each of the at most two children of y and of at most $k+1$ vertices from y itself, a total of at most $2b+k-1 \leq n_{\max}$ vertices. This establishes properties 1–3 of the cluster partition.

For $i = 1, \ldots, s$, let C_i' be the set of those nodes in C_i that are adjacent in T to a node not in C_i, take $Z_i = \bigcup_{x\in C_i'} U_x$, and let H_i be the graph induced by the vertices in $(\bigcup_{x\in C_i} U_x)\backslash Z_i$. By the properties of tree decompositions, the graphs $H_1, \ldots, H_s$ are disjoint, and the boundary of H_i is contained in Z_i for $i = 1, \ldots, s$.

For $i = 1, \ldots, s$, we will call H_i a *kernel* if $|C_i'| \leq 2$ and C_i is not the cluster containing the root of T (we exclude the latter cluster because of its special status). We next establish a lower bound on the number of kernels.

CLAIM 2. *The number of kernels is at least* $\frac{1}{2}n/n_{max}$.

Proof. By property 2 of the cluster partition and the fact that $\bigcup_{i=1}^{s}\bigcup_{x\in C_i} U_x = V$, the number s of clusters is at least $n/n_{\max}$. The clusters form a *cluster tree* in a natural fashion: two clusters are adjacent if one contains a node adjacent in T to a node in the other cluster, and the degree of a cluster C_i is at least $|C_i'|$ (it exceeds $|C_i'|$ if some node in C_i' has more than one neighbor outside of C_i). In general, a tree with m nodes contains $m-1$ edges. Hence if h denotes the number of nodes of degree ≥ 3 in an m-node tree and $m \geq 2$, we have $3h + (m-h) \leq 2(m-1)$ and thus $h \leq m/2 - 1$. Applying this to the cluster tree, with at least $n/n_{\max} > 1$ nodes, only one of which contains the root of T, shows that the number of clusters of degree ≤ 2 is at least $1 + \frac{1}{2}n/n_{\max}$, and hence that the number of kernels is at least $\frac{1}{2}n/n_{\max}$. □

The boundary of a kernel H contains at most $2(k+1)$ vertices and, by property 2 of the cluster partition, H contains no more than $n_{\max}$ vertices. In particular, since $d \geq n_{\max} + 2(k+1)$, all vertices in H are small. It follows that every weakly connected component of H is a valley. We will call a valley of this kind *good* if it intersects a plain in $\mathcal{P}$, and *bad* otherwise. Note that a bad valley contains fewer than $n_{\min}$ vertices (otherwise it would be a plain, contradicting the maximality of $\mathcal{P}$). A bad valley with boundary B will be called a B-valley.

We classify the bad valleys into three types depending on their boundaries. Consider a bad valley L with boundary B. If B contains one or more small vertices, L is of type a. If B contains only large vertices and $B \not\subseteq U_x$ for all $x \in X$, L is of type b. If B contains only large vertices and $B \subseteq U_x$ for some $x \in X$, finally, L is of type c. We next bound the number of valleys of type a per kernel, the number of kernels containing valleys of type b, and the total number of valleys of type c.

CLAIM 3. *The number of valleys of type* a (B *contains a small vertex*) *in any fixed kernel is bounded by* $2(k+1)$.

Proof. Every valley of type a "uses up" one or more of the at most $2(k+1)$ boundary vertices of its kernel. Here we use the fact that valleys need only be weakly connected: two vertices in the same kernel and with a common small neighbor belong to the same valley. □

We now consider the B-valleys for which B contains only large vertices. Given such a B-valley L, choose $v \in B$ such that the root r_v of the subtree T_v of T induced by the node set $\{x \in X \mid v \in U_x\}$ is of maximal depth, call v the *low point* of L, and assign L (for the purpose of accounting) to its low point. We here use the fact that G is connected, which ensures that $B \neq \emptyset$.

CLAIM 4. *The total number of kernels containing one or more valleys of type* b ($B \not\subseteq U_x$ *for all* $x \in X$) *is bounded by* $(2k/d) \cdot n$.

Proof. Let L be a bad valley and let B and v be its boundary and its low point, respectively. We can conclude from the root-seeking principle that r_v lies within the cluster C containing L; otherwise B would be contained in U_x, where x is the node in C of minimal depth in T. Since v is large, the number of clusters containing valleys of type b is therefore bounded by the number of large vertices. Because G has at most kn edges, the latter number in turn is bounded by $(2k/d) \cdot n$. ∎

CLAIM 5. *The total number of valleys of type* c ($B \subseteq U_x$ *for some* $x \in X$) *is bounded by* $(2^{k+2}k \cdot n_{min}/d) \cdot n$.

Proof. Let v be a large vertex in V and let B be a subset of V. By definition, all B-valleys are twins. Thus no d consecutive entries in the adjacency list of v can contain entries of vertices in $n_{\min}$ or more different B-valleys, since then all of these (bad) B-valleys would be acquainted, and some of them (with a suitable total size) would form a plain, contradicting the maximality of $\mathcal{P}$. We may conclude that at most $\lceil \deg(v)/d \rceil \cdot n_{\min} \leq 2 \deg(v) \cdot n_{\min}/d$ B-valleys are assigned to v, where $\deg(v)$ denotes the degree of v and the inequality follows from the fact that $\deg(v) > d$.

The valleys assigned to v may not all be twins. However, the root-seeking principle ensures that if a B-valley is assigned to v and B is contained in some (single) bag, then $B \subseteq U_{r_v}$. Thus the valleys of type c assigned to v have at most 2^k different boundaries (all such boundaries are subsets of a fixed set of at most $k+1$ vertices, and all contain v). It follows that the total number of valleys of type c assigned to v is bounded by $2^{k+1} \deg(v) \cdot n_{\min}/d$. Again since the total number of edges is at most kn, this sums over all vertices v to at most $(2^{k+2}k \cdot n_{\min}/d) \cdot n$. ∎

Since a bad valley contains fewer than $n_{\min}$ vertices and a kernel contains at least $b - 2(k+1)$ vertices, each kernel containing only bad valleys decomposes into at least $(b - 2(k+1))/n_{\min} \geq 3(k+1)$ bad valleys. At most $2(k+1)$ of these are of type a (Claim 3). Hence if a kernel contains only bad valleys, then either one or more of these are of type b, or at least $k+1$ of them are of type c. The first condition applies to at most $(2k/d) \cdot n \leq \frac{1}{8} n/n_{\max}$ kernels (Claim 4), and because the total number of valleys of type c is bounded by $(2^{k+2}k \cdot n_{\min}/d) \cdot n$ (Claim 5), the second condition applies to at most $(2^{k+2}k \cdot n_{\min}/(d(k+1))) \cdot n \leq \frac{1}{4} n/n_{\max}$ kernels. Since the total number of kernels is at least $\frac{1}{2} n/n_{\max}$ (Claim 2), at least $(\frac{1}{2} - \frac{1}{4} - \frac{1}{8}) \cdot n/n_{\max} = \frac{1}{8} n/n_{\max}$ kernels contain one or more good valleys. At most $n_{\max}$ disjoint valleys can intersect the same plain, so the number of plains in $\mathcal{P}$ is at least cn if we take $c = 1/(8 \cdot n_{\max}^2)$. This ends the proof of Lemma 3.1. ∎

4. Constructing tree decompositions. In this section we show that minimum-width tree decompositions of n-vertex graphs of bounded treewidth can be constructed on an EREW PRAM using $O((\log n)^2)$ time and $O(n)$ operations. More precisely, given an n-vertex graph G and a constant k, our algorithm outputs either a tree decomposition of G of treewidth $tw(G)$ or an indication of the fact that $tw(G) > k$.

The algorithm is based on the graph-reduction technique: a connected input graph of treewidth $\leq k$ is successively replaced by smaller and smaller graphs in a

series of *reductions* until a constant-size graph results. Starting from a minimum-width tree decomposition of the final constant-size graph, the reductions are then undone one by one in the reverse order of their application, where, in undoing a reduction that originally replaced a graph G' by a smaller graph G'', a minimum-width tree decomposition of G' is derived from one of G''. At the end of this process we obtain a minimum-width tree decomposition of the input graph.

Suppose that v and w are vertices in a graph G' that are either adjacent or twins and let G'' be the graph obtained from G' by removing v and its incident edges after first inserting an edge between w and each neighbor of v that was not previously a neighbor of w; we will call v and w *reduction partners* and say that G'' is obtained from G' by reduction *on* the pair $\{v, w\}$. A tree decomposition of G'' can be obtained from any tree decomposition of G' by replacing each occurrence of v in a bag by w if v and w are adjacent in G' and by removing all occurrences of v if v and w are twins in G'; hence $tw(G'') \leq tw(G')$. On the other hand, $tw(G') \leq tw(G'') + 1$, since a tree decomposition of G' can be obtained from any tree decomposition of G'' by replacing each occurrence of w in a bag by occurrences of both v and w—we will say that w is *expanded*. If G' is of bounded treewidth, we can therefore undo the reduction transforming G' into G'' by applying the width-minimizing procedure of Theorem 2.3 to derive a minimum-width tree decomposition of G' from one of G''.

For a fast parallel algorithm it clearly does not suffice to remove vertices one by one. It is easy to see, however, that the scheme described in the preceding paragraph remains valid if, rather than reducing on a single pair of vertices, we reduce simultaneously on an arbitrary collection of pairs that are sufficiently far apart in the graph not to interfere with each other. The only difference is that the treewidth of G' may now be as much as twice that of G'', plus one (each vertex in a bag may need to be expanded into two vertices), which is still fine for the width-minimizing procedure.

THEOREM 4.1. *For all constants $k \geq 1$ and all integers $n \geq 2$, the following problem can be solved on an EREW PRAM using $O((\log n)^2)$ time, $O(n)$ operations, and $O(n)$ space*: *given an n-vertex graph G, construct a minimum-width tree decomposition of G or decide* (*correctly*) *that $tw(G) > k$.*

Proof. For the time being assume that G is connected and of treewidth at most k. We will apply Lemma 3.1 to G with $n_{\min} = 2$. Hence let the constants c and d be as in the lemma and define the concepts *small* and *acquainted* accordingly. The lemma implies that G contains at least $cn/2$ distinct pairs $\{v, w\}$ of small vertices such that v and w are either adjacent or acquainted twins; to see this, note that each of the cn disjoint plains whose existence is guaranteed by the lemma contains either a valley of at least two vertices, hence a small vertex with a small neighbor (either the small neighbor also belongs to the plain or it is one of its boundary vertices), or two or more acquainted twin valleys of one vertex each. Furthermore, the set R of all such pairs can be computed in constant time using $O(n)$ operations, since it suffices to let each small vertex inspect all its neighbors and all vertices with which it is acquainted; with some care, this can be done without concurrent reading.

We cannot necessarily execute all reductions corresponding to pairs in R, since vertices in distinct pairs may coincide, be adjacent, or have adjacent entries in some adjacency list, which hinders the simultaneous execution of the associated reductions. In order to deal with this complication, we construct a *conflict graph* with a vertex for each pair in R and an edge between two vertices if the corresponding reductions exclude each other for one of the reasons mentioned above. It is easy to see that the conflict graph is of bounded degree and can be constructed in constant time using

$O(n)$ operations. Following [21], we define a *fractional independent set* in an m-vertex graph H as an independent vertex set in H of size at least ϵm, where ϵ is an (unspecified) positive constant. We proceed to compute a fractional independent set in the conflict graph, which can be done in $O(\log n)$ time using $O(n)$ operations [28, Lemma 7(b)]. Finally we execute the reductions on the pairs in the independent set, which takes constant time and uses $O(n)$ operations.

The reductions described above change G into a smaller graph G'. Let us now see that we can undo the reductions in the sense of deriving a minimum-width tree decomposition of G from one of G'. We already observed that all that is involved is to expand certain vertices into the corresponding pair of reduction partners, after which we can finish using the width-minimizing procedure of Theorem 2.3. Allowing concurrent reading, the task would be trivial—processors collectively inspecting the whole tree decomposition could simply expand each such vertex after looking up its partner in a table. In order to avoid concurrent reading from the table, we begin by balancing the given tree decomposition of G' (Lemma 2.2); this may increase its width but only by a constant factor. We then process the resulting balanced tree decomposition $(T = (X, F), \{U_x \mid x \in X\})$ in topological order; i.e., each node is processed before all of its children. The processing of a node x in T expands all vertices in U_x that need to be expanded. If x is the root of T, this is easy. If not, the identity of the reduction partner of each relevant vertex $v \in U_x$ can be passed to x from its parent y, except if v occurs in U_x for the first time (i.e., if $v \notin U_y$). For each vertex v the latter happens only at a single tree node x, however, so that in this case we can use table lookup to find the reduction partner of v without any risk of concurrent reading. The balanced tree decomposition can be processed as described in $O(\log n)$ time using $O(n)$ operations.

The graph G' derived from G is connected and of treewidth at most k, so that a new batch of reductions can be applied to G'. Since G' is smaller than G by a constant factor, as measured by the number of vertices, $O(\log n)$ successive stages of simultaneous reductions suffice to reduce the input graph to a graph of constant size. Provided that the representation of the graph at hand is compacted after each stage by means of prefix summation, the number of operations and the space needed decrease geometrically over the stages, so that the whole process uses $O((\log n)^2)$ time, $O(n)$ operations, and $O(n)$ space. Undoing the reductions is no more expensive. This proves Theorem 4.1 for connected input graphs of treewidth at most k.

Suppose now that the input graph G is of treewidth at most k but not connected. Our approach will be to apply the algorithm developed above not to G but to an auxiliary connected graph H obtained from G by introducing a new vertex r and an edge between r and a single vertex in each connected component of G. Except in the trivial case in which G has no edges, G and H have the same treewidth, so that a minimum-width tree decomposition of G can be obtained from a minimum-width tree decomposition of H by removing the occurrences of r from all bags. In order to select a vertex from each connected component of G, we can apply the first part of the reduction algorithm to G in a preprocessing phase: each connected component of G, being of treewidth at most k, is reduced to constant size, at which point the selection is easy, and the component can be removed (since its size may not decrease any further, keeping it around might make subsequent stages too expensive).

If the treewidth of the input graph G is larger than k, one or more of its connected components may fail to be reduced to constant size within the time bound established for graphs of treewidth at most k, in which case the algorithm can stop and announce

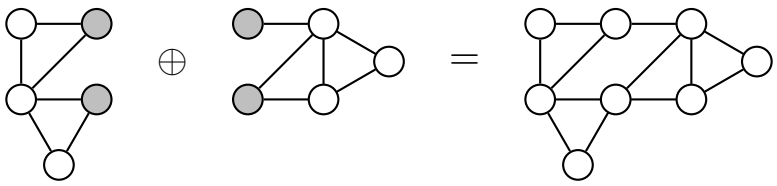


FIG. 2. *Combination of two terminal graphs using* $\oplus$.

that $tw(G) > k$. It is easy to see from the description of the algorithm that even if $tw(G) > k$, the algorithm never performs an illegal action such as concurrent reading, and any output produced by the algorithm is a correct minimum-width tree decomposition of G. □

By applying first the algorithm of Theorem 4.1 and then that of Theorem 2.5, we obtain the result below.

COROLLARY 4.2. *For all constants $k \geq 1$ and all integers $n \geq 2$, the following problem can be solved on an EREW PRAM using $O((\log n)^2)$ time, $O(n)$ operations, and $O(n)$ space*: *given an n-vertex graph G, construct a minimum-width path decomposition of G or decide* (*correctly*) *that the pathwidth of G is larger than k.*

5. Deciding treewidth on the EREW PRAM. An important bottleneck for the running time of the algorithm in the previous section is the repeated application of the algorithm of Theorem 2.3 while undoing the reductions. When we aim for a decision algorithm only, we can follow a different approach: we will not undo reductions, but instead make sure that all reductions preserve treewidth. We actually describe a generic algorithm, whose instantiations solve various decision problems on graphs of bounded treewidth; in the more general setting, reductions must not affect membership in the class of graphs to be recognized.

Our algorithm can be viewed as a parallelization of a linear-time sequential algorithm due to Arnborg et al. [6]. A first parallel version of this algorithm was given in [9]. The algorithm described there is randomized, works only for graphs of bounded degree, and uses $O(\log n)$ expected time and $O(n \log n)$ expected operations on n-vertex input graphs. The algorithm given in this section works for arbitrary graphs, uses $O(n)$ operations, and is deterministic but at a cost of an extra factor of $O(\log^* n)$ in the running time. The algorithm of [6] uses an amount of space bounded by a polynomial but a polynomial whose degree is large and unspecified. We reduce this to $O(n)$ by means of the bounded adjacency-list search technique.

A *terminal graph* is a triple $G = (V, E, Z)$, where (V, E) is a graph and $Z \subseteq V$ is an ordered set of distinguished vertices in G. The vertices in Z and those in $V \backslash Z$ are called the *terminals* and the *internal vertices* of G, respectively. A terminal graph is *open* if there are no edges between terminals. For $l \geq 0$, an *l-terminal graph* is a terminal graph with exactly l terminals. Let $\mathcal{H}_l$ denote the class of l-terminal graphs for $l \geq 0$.

Given two l-terminal graphs G_1 and G_2 for some $l \geq 0$, we define $G_1 \oplus G_2$ as the graph obtained by taking the disjoint union of G_1 and G_2 and then identifying the ith terminals in G_1 and G_2 for $i = 1, \ldots, l$. An example is shown in Figure 2. When there is an edge between a pair of terminals in both G_1 and G_2, we take just a single edge between these in $G_1 \oplus G_2$.

Let $\mathcal{G}$ be a class of graphs. We define an equivalence relation $\sim_{\mathcal{G}}$ on the set of terminal graphs as follows: $G_1 \sim_{\mathcal{G}} G_2$ if and only if for some l, G_1 and G_2 both have

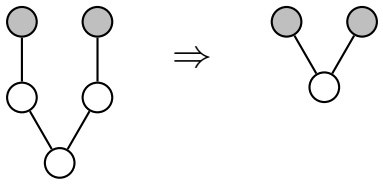


FIG. 3. *A reduction that preserves (non)membership in* $\mathcal{C}^2$.

l terminals, and for all $H \in \mathcal{H}_l$, we have $G_1 \oplus H \in \mathcal{G}$ if and only if $G_2 \oplus H \in \mathcal{G}$. Informally, G_1 and G_2 are equivalent under $\sim_{\mathcal{G}}$ if any occurrence of G_1 in a bigger graph can be replaced by an occurrence of G_2 without affecting membership of the bigger graph in $\mathcal{G}$. We say that a class $\mathcal{G}$ and its defining property P (i.e., $G \in \mathcal{G}$ if and only if $P(G)$) are of *finite index* if, for every $l \geq 0$, $\mathcal{H}_l$ splits into a finite number of equivalence classes under $\sim_{\mathcal{G}}$. (Graph properties of finite index are also known as being *regular* or of *finite state*.) Many important properties are known to be of finite index.

We illustrate the concepts introduced above through a simple example, in which P denotes the property of being a simple cycle of even length and $\mathcal{G}$ is the class $\mathcal{C}^2$ of graphs with the property P. $\mathcal{H}_2$, the class of 2-terminal graphs, splits into seven equivalence classes $\mathcal{H}_2^{(0)}, \ldots, \mathcal{H}_2^{(6)}$ under the relation $\sim_{\mathcal{C}^2}$. $\mathcal{H}_2^{(0)}$ is the class of 2-terminal graphs without edges and without internal vertices. $\mathcal{H}_2^{(1)}$, $\mathcal{H}_2^{(2)}$, and $\mathcal{H}_2^{(3)}$ are classes of 2-terminal graphs consisting of a simple path between the two terminals. The length of this path is 1 in the case of $\mathcal{H}_2^{(1)}$, is even for $\mathcal{H}_2^{(2)}$, and is odd and at least 3 for $\mathcal{H}_2^{(3)}$. $\mathcal{H}_2^{(4)}$ and $\mathcal{H}_2^{(5)}$ are classes of 2-terminal graphs that are cycles of even length. The two terminals are adjacent in graphs in $\mathcal{H}_2^{(4)}$ but not in those in $\mathcal{H}_2^{(5)}$. $\mathcal{H}_2^{(6)}$, finally, is the class of all remaining 2-terminal graphs, such as those containing a vertex of degree 3 or more. The reader may want to determine the set A of those pairs $(l_1, l_2) \in \{0, \ldots, 6\}^2$ such that if $G_1 \in \mathcal{H}_2^{(l_1)}$ and $G_2 \in \mathcal{H}_2^{(l_2)}$, then $G_1 \oplus G_2 \in \mathcal{C}^2$. E.g., $(2, 2) \in A$, but $(2, 3) \notin A$.

The two 2-terminal graphs shown in Figure 3 are equivalent under the relation $\sim_{\mathcal{C}^2}$ (both belong to $\mathcal{H}_2^{(2)}$). Thus if a graph G' is obtained from another graph G by replacing an occurrence of the left-hand graph in Figure 3 by an occurrence of the right-hand graph, we know that $G' \in \mathcal{C}^2$ if and only if $G \in \mathcal{C}^2$. This reduction allows us to recognize the graphs in $\mathcal{C}^2$: when $G \in \mathcal{C}^2$, we can apply the reduction repeatedly until we are left with a cycle of constant size. The recognition can also be carried out in parallel by applying many reductions simultaneously.

As we will now show, the ideas developed in the previous paragraph extend to any situation in which a graph class of interest can be characterized by an assertion of the form $P(G) \wedge (tw(G) \leq k)$, where P is a graph property of finite index and k is a fixed integer (in the example above, any $k \geq 2$ will do).

THEOREM 5.1. *For every graph property P of finite index and for all constants $k \geq 1$ and all integers $n \geq 2$, the problem of deciding whether $P(G) \wedge (tw(G) \leq k)$ for an n-vertex input graph G can be solved on an EREW PRAM using $O(\log n \log^* n)$ time, $O(n)$ operations, and $O(n)$ space.*

Proof. Assume first that $P(G)$ implies that G is connected. It was shown in [36] that the class of graphs of treewidth at most k is of finite index, and one easily observes that the conjunction of two properties of finite index is again of finite index

(see, e.g., [14]). Hence $\mathcal{G} = \{G \mid P(G) \wedge (tw(G) \leq k)\}$ is of finite index. Let $\mathcal{R}$ be a finite set of open terminal graphs that contains at least one element of each equivalence class of $\sim_{\mathcal{G}}$ comprising one or more open terminal graphs with at most $2(k+1)$ terminals, and take $n_{\min}$ as one more than the largest number of vertices of any graph in $\mathcal{R}$. By Lemma 3.1, we can choose constants $d, n_{\max} \geq 1$ and $c > 0$ such that if G has more than $n_{\max}$ vertices and $G \in \mathcal{G}$, then G contains at least cn disjoint $(d, k, n_{\min}, n_{\max})$-plains (for any adjacency-list representation of G).

The significance of $n_{\min}$ is that any open terminal graph with at least $n_{\min}$ vertices and at most $2(k+1)$ terminals has a smaller equivalent terminal graph in $\mathcal{R}$. In particular, each plain H together with its boundary B and all edges joining a vertex in H and a vertex in B, with the vertices in B considered as terminals (call this an *extended plain*), is such an open terminal graph, so that it can be replaced by a smaller terminal graph in $\mathcal{R}$. Considering isomorphic graphs as identical, there is only a finite number of different extended plains, all of which can therefore be mapped to equivalent smaller open terminal graphs by means of a finite table T. Each entry in T corresponds to a reduction in a natural way.

The algorithm proceeds in a number of phases. In each phase, each vertex determines whether it belongs to a plain and, if so, looks up a corresponding reduction in T. This can be done in constant time: it suffices to let each vertex u inspect those vertices and edges that lie on a path of length at most $2n_{\max}$ from u such that the entries of any two consecutive edges (v, w) and (w, x) on the path are separated by a distance of at most d in the adjacency list of w; this can be done without concurrent reading. The reductions found by two distinct vertices may not be simultaneously executable: the plain containing one vertex may intersect the plain containing the other vertex or its boundary, or two vertices, one from each plain, may have adjacent entries in some common boundary vertex. Because we only replace open terminal graphs by other open terminal graphs, however, these are the only ways in which two reductions can interfere with each other. As in section 4, we construct a conflict graph of bounded degree on the vertices belonging to plains, compute a fractional independent set in the conflict graph, and execute the corresponding reductions, which reduces the size of the graph by at least a constant factor. After $O(\log n)$ stages, either we are left with a graph of constant size, whose membership in $\mathcal{G}$ can be decided directly, or the input graph did not belong to $\mathcal{G}$.

The only part of a stage that takes more than constant time with a linear number of processors is the computation of a fractional independent set in the conflict graph. For this, we employ in the first $O(\log^* n)$ stages the algorithm of [28, Lemma 7(b)], which uses $O(\log m)$ time and $O(m)$ operations, where m is the number of vertices in the conflict graph. In the remaining phases, we use the algorithm of [26, Theorem 4], which needs $O(\log^* n)$ time and $O(m \log^* n)$ operations. The total time is $O(\log n \log^* n)$, and a simple simulation argument that schedules compactions of the representation conveniently (see [30, section 4]) shows that the algorithm can be carried out using $O(n)$ operations.

Dropping the assumption that $P(G)$ implies that G is connected, we can still proceed as described above, provided that we remove and save each connected component with fewer than $n_{\min}$ vertices as soon as it arises. After $O(\log n)$ stages, either the input graph has been reduced to an equivalent collection of connected graphs, each of which contains fewer than $n_{\min}$ vertices, or it did not belong to $\mathcal{G}$. Assume the former. In constant time, a single processor can *combine* two graphs in the collection, i.e., replace them by a single graph equivalent to their union and containing

fewer than $n_{\min}$ vertices. By means of a tree-structured combination process that uses $O(\log n)$ time and $O(n)$ operations, we can therefore reduce the input graph to a single equivalent graph with fewer than $n_{\min}$ vertices, for which membership in $\mathcal{G}$ can be decided directly. □

The theorem implies, in particular, that the problem of deciding whether the treewidth of a given graph is at most k, for constant k, can be solved in $O(\log n \log^* n)$ time with $O(n)$ operations. Moreover, the same result can be shown to hold for pathwidth. Many well-known graph properties are of finite index. For instance, this is true of all properties that can be expressed in monadic second-order logic, such as Hamiltonicity and l-colorability. This was first shown by Courcelle [20]; see [14] for a possibly more accessible proof.

Theorem 5.1 is nonconstructive: an algorithm with the stated properties is merely shown to exist. To actually exhibit the algorithm, we must be able to compute the number $n_{\min}$ and to construct the table T. If we have a terminating algorithm that decides whether two given terminal graphs are equivalent under $\sim_{\mathcal{G}}$ or under some refinement (subdivision) of $\sim_{\mathcal{G}}$ that still has a finite number of equivalence classes, this can be done by a method described in [6] (in a general algebraic setting). For the case in which $\mathcal{G}$ is the class of all graphs of treewidth at most k, such an explicit decision algorithm was exhibited in [36]. If $\mathcal{G}$ is the set of those graphs of treewidth at most k that satisfies a property P expressed in monadic second-order logic, then an algorithm that decides a subdivision of $\sim_{\mathcal{G}}$ with a finite number of equivalence classes can be obtained by combining results implicit in [14, 20, 36].

It is also possible to apply the parallel reduction techniques to problems that are of *finite integer index*, in the sense of [9]. This allows deciding on the size of a maximum independent set, minimum vertex cover, minimum dominating set, and others on graphs of bounded treewidth in $O(\log n \log^* n)$ time using $O(n)$ operations on an EREW PRAM. Using the technique of [9, section 6.1], it is also possible to construct corresponding solutions for some of these problems. With this technique and a more refined analysis, Bodlaender and de Fluiter obtained algorithms that work on an EREW PRAM using $O(\log n \log^* n)$ time and $O(n)$ operations for the problems of recognizing and building a parse tree of series-parallel graphs and of constructing a minimum-width tree decomposition of graphs of treewidth 2 [12, 23].

6. Deciding treewidth on the CRCW PRAM. In this section we show how the running time of $O(\log n \log^* n)$ of the algorithm in the previous section can be reduced to $O(\log n)$ if we move to the stronger CRCW PRAM. Among the many variants of the CRCW PRAM, we employ one that allows m processors to compute the OR of m bits in constant time using $O(m)$ space for all integers $m \geq 1$; this requirement excludes none of the CRCW PRAM variants commonly considered. We assume an instruction set that includes unit-time binary left and right shifts of words of $O(\log n)$ bits by amounts specified in a second word.

As concerns its running time, the EREW PRAM algorithm has two bottlenecks: first, as dictated by efficiency considerations, the representation of the graph at hand is compacted $\Theta(\log^* n)$ times, with each compaction taking logarithmic time. Second, in each of $\Theta(\log n)$ stages a fractional independent set is found in a conflict graph of bounded degree for which we spend $\Theta(\log^* n)$ time per stage. Moving to the CRCW PRAM, we can easily eliminate the first bottleneck, since in this model compaction can be done in $\Theta(\log n / \log\log n)$ time [19] rather than the $\Theta(\log n)$ time for the EREW PRAM. Before attacking the second bottleneck, let us observe that we can execute $\Theta(\log n / \log^* n)$ stages of the EREW PRAM algorithm without exceeding a

time bound of $O(\log n)$. After compacting once, we can then associate $2^{\Omega(\log n/\log^* n)}$ processors with each remaining vertex in the graph, We will express this by saying that we have a *processor advantage* of $2^{\Omega(\log n/\log^* n)}$, which is much more than what we need in the following.

The remaining problem is to finish the computation in $O(\log n)$ time making use of the large processor advantage mentioned above, which we will do by means of derandomization. The task is, for a positive integer $m \leq n$, to compute a fractional independent set I in an m-vertex graph of bounded degree in constant time. Observe that there is a very simple randomized algorithm for obtaining I: each vertex picks a random bit uniformly from $\{0, 1\}$ and independently of other vertices and then steps into I exactly if it picked a 1 while each of its neighbors picked a 0. Although this formulation assumes that the vertices make independent choices, it is easy to see that much less will also do. If each vertex v has at least a constant probability of stepping into I, then the expected size of I is $\Omega(m)$, so that, obviously, at least one possible execution of the randomized algorithm will result in $|I| = \Omega(m)$. Whether v steps into I, however, is a function only of the random bits picked by v and by its neighbors; i.e., it suffices to guarantee d-wise independence, where d is one more than the maximum degree of the graph. In the case of perfect d-wise independence, the probability that v steps into I is at least 2^{-d}. Since we can allow any positive constant here instead of 2^{-d}, however, we can relax the requirements even more. For $\epsilon > 0$, random bits $X_1, \ldots, X_m$ are said to be (ϵ, d)-*independent* [3] if, for all positive integers $l \leq d$, all distinct integers $i_1, \ldots, i_l$ with $1 \leq i_1, \ldots, i_l \leq m$ and all $b_1, \ldots, b_l \in \{0, 1\}$, the probability of the event $X_{i_1} = b_1, X_{i_2} = b_2, \ldots, X_{i_l} = b_l$ deviates from 2^{-l} by at most ϵ (d-wise independence is the special case $\epsilon = 0$). It is easy to see that (ϵ, d)-independent random bits, where $\epsilon = 2^{-d-1}$, suffice for our purpose. We now appeal to Theorem 2 of [3], which promises that m $(2^{-d-1}, d)$-independent random bits can be drawn from a sample space of size $(\log m)^{O(1)}$ (where the exponent depends on d); we argue separately in Lemma 6.2 below that the computation of the m bits can be carried out in constant time with m processors.

Since our processor advantage is much bigger than polylogarithmic in n, we can use the limited-randomness algorithm developed above and simulate all $(\log m)^{O(1)} = (\log n)^{O(1)}$ possible executions of it in parallel. We know that at least one execution will be good, in the sense that it will lead to an independent set I of size $\Omega(m)$. We would like simply to pick a good execution, but this is not entirely trivial, since we have only constant time per stage, which is not sufficient for computing the size of I. Using the deterministic approximate-summation algorithm of [29, Theorem 3], we can compute the size of I, up to a constant factor (which is sufficiently accurate), in $O((\log \log n)^3)$ time. While this is fast, it is not fast enough. We overcome this using a technique of [29], namely to simulate all possible executions of the randomized algorithm not just for one stage at a time but for $\Theta((\log \log n)^3)$ consecutive stages, after which we can spend $O((\log \log n)^3)$ time determining a good execution without violating our time bound (as much time is then spent on graph reduction as on counting). Doing this increases the size of the sample space to $(\log n)^{O((\log \log n)^3)} = 2^{O((\log \log n)^4)}$, which is still sufficiently small, in view of our larger processor advantage.

THEOREM 6.1. *For every graph property P of finite index and for all constants $k \geq 1$ and all integers $n \geq 2$, the problem of deciding whether $P(G) \wedge (tw(G) \leq k)$ for an n-vertex input graph G can be solved on a CRCW PRAM using $O(\log n)$ time, $O(n)$ operations, and $O(n)$ space.*

As in the case of Theorem 5.1, Theorem 6.1 is nonconstructive; see the discussion

near the end of section 5. In order to complete the proof of Theorem 6.1, we still have to show the following.

LEMMA 6.2. *For all given integers $m, K \geq 2$ with $K = (\log m)^{O(1)}$ and all constant integers $d \geq 2$, m $(1/K, d)$-independent random bits can be computed in constant time on a CRCW PRAM using m processors, $O(m)$ space, and a single random integer drawn from the uniform distribution over a range of size $(\log m)^{O(1)}$.*

Proof. Our construction, described below, is an elaboration of one given in [3, Theorem 2].

Without loss of generality assume that d is odd, say, $d = 2t + 1$. Let r be the smallest number of the form $2 \cdot 3^i$ no smaller than $\log(m + 1)$, where i is an integer, and take p as the smallest prime no smaller than $(2K(1 + rt))^2$.

Let F be the set of all bit vectors of length r and denote by $\phi : \{0, \ldots, 2^r - 1\} \to F$ the function that maps each integer to its standard r-bit binary representation. Assume that F is organized into a field by means of suitable addition and multiplication operations; details will be given below.

Now choose a random integer h from the uniform distribution over $\{0, \ldots, p - 1\}$ and compute the bit vector y of length $1 + rt$ whose $(i + 1)$st bit, for $i = 0, \ldots, rt$, is 0 exactly if $i + h \equiv s^2 \pmod p$ for some $s \in \{1, \ldots, p - 1\}$ (i.e., if $h + i$ is a quadratic residue modulo p). Finally, for $i = 1, \ldots, m$, compute the ith output bit as the inner product modulo 2 of y with a bit vector x_i of length $1 + rt$ constructed as follows: the first bit of x_i is 1, the next r bits are those of $\phi(i)$, the next r bits are those of $(\phi(i))^3$, where the powering is done according to the multiplication in F, the next r bits are those of $(\phi(i))^5$, etc., until the last r bits, which are those of $(\phi(i))^{2t-1}$.

It is proved in [3, Proposition 2] that the inner product modulo 2 of y with any fixed nonzero bit vector of length $1+rt$ is ϵ-*biased* [3], where $\epsilon = rt/\sqrt{p}+(1 + rt)/p \leq 2(1 + rt)/\sqrt{p} \leq 1/K$; i.e., it takes on the values 0 and 1 with probabilities differing by at most ϵ. It then follows from [3, Lemma 2] and [2, Proposition 6.5] that the m bits output by the algorithm are indeed $(1/K, d)$-independent (a requirement of [3, Lemma 2] to the effect that the size of a sample space called S_n^m must be a power of 2 is not satisfied in the present application, where this size is the prime p, but is in fact superfluous). What remains is to bound the resources needed by the computation.

Obviously, $r = O(\log m)$ and, by Bertrand's postulate (see, e.g., [32, Theorem 418]), which asserts the existence of a prime in the range $\{s, \ldots, 2s\}$ for every positive integer s, we obtain that $p = (\log m)^{O(1)}$, so that the random integer h is indeed chosen from a range as small as claimed in the lemma. In order to compute r and p in constant time, we proceed as follows: first, comparing the powers $2^1, 2^2, 2^3, \ldots$ with $m + 1$ in parallel, we obtain $\lceil \log(m + 1) \rceil$. It is then easy to compute an integer g with $g \geq p$, but $g = (\log m)^{O(1)}$. The remaining computation uses g^2 processors. Using trial division by all smaller numbers in the set $S = \{2, \ldots, g\}$, we determine the set of primes in S. Subsequently, using trial division by all smaller primes, we determine the set of powers of 3 in S, after which it is easy to select first r and then p; each is the smallest number in S satisfying a property that can be tested in constant time by one processor. The vector y is obtained similarly by first computing the set of quadratic residues modulo p.

In order to construct the vectors $x_1, \ldots, x_m$, we need to implement the field operations of F. We define the product of the vectors $(a_{r-1}, a_{r-2}, \ldots, a_0)$ and $(b_{r-1}, b_{r-2}, \ldots, b_0)$ as $(c_{r-1}, c_{r-2}, \ldots, c_0)$, where $\sum_{i=0}^{r-1} c_i x^i$ is the remainder polynomial obtained by dividing the product $(\sum_{i=0}^{r-1} a_i x^i)(\sum_{i=0}^{r-1} b_i x^i)$ by the fixed polynomial $f(x) = x^r + x^{r/2} + 1$ over the 2-element field $\mathbb{Z}_2$. Since $f(x)$ is irreducible over $\mathbb{Z}_2$ [37,

Exercise 3.96], it is well-known that this multiplication operation together with componentwise addition over $\mathbb{Z}_2$ indeed turns F into a field.

Compute q as a positive integer with $q \leq (\log m)/5$, but $q = \Omega(\log m)$. We can implement addition and multiplication over $\mathbb{Z}_2$ of polynomials of degree less than q, represented by bit vectors in the obvious way, by means of table lookup. In each case, we need a $(2^q - 1) \times (2^q - 1)$ table with entries in the range $\{0, \ldots, 2^{2q-1} - 1\}$. In constructing the tables, we can therefore, for each table entry and each integer in the range $\{0, \ldots, 2^{2q-1} - 1\}$, dedicate a team of $\Omega(m^{1/5})$ processors to testing whether the integer is the correct value of the entry under consideration, in which case the team will fill in that table entry. In the case of addition, the testing is trivial to do in constant time with just q processors, each of which takes care of one bit position. For multiplication, the problem reduces to computing the parities of $2q-1$ bit sequences, each of length at most q. Since the parity of q bits can be computed in constant time with $O(m^\delta)$ processors, for arbitrary constant $\delta > 0$ (see, e.g., [40, lemma, p. 375]), we have enough processors in this case as well.

Because $r = O(q)$, addition and multiplication over $\mathbb{Z}_2$ of polynomials of degree less than r reduces to a constant number of additions and multiplications over $\mathbb{Z}_2$ of polynomials of degree less than q, so that both operations can be carried out in constant time by one processor using the tables constructed above. In order to complete the implementation of multiplication over F, we need to describe how to compute the remainder over $\mathbb{Z}_2$ of a polynomial $a(x) = \sum_{i=0}^{l} a_i x^i$ modulo $f(x)$, where $r \leq l \leq 2r - 2$. But since none of the powers $x^{r-1}, \ldots, x^{(r/2)+1}$ occur in $f(x)$, the polynomial $a(x) - (\sum_{i=r}^{l} a_i x^{i-r}) f(x)$, which obviously has the same remainder modulo $f(x)$ as $a(x)$, is of degree at most $\max\{l - r/2, r - 1\}$, so that constant time suffices to reduce the degree of the input polynomial by at least $r/2$ or below r. Doing this twice completes the computation.

The final operation that must be supported is forming the inner product modulo 2 of two bit vectors, each of length $O(r)$. This operation can easily be carried out in constant time by one processor using a table that maps each bit sequence of q bits to its parity. Before the table can be used, it is necessary to convert y from a representation with one bit per word to one with q bits per word. Again, this can be done in constant time by trying out all possibilities in parallel. □

Acknowledgments. We thank Jordan Gergov and Rajeev Raman for helpful discussions related to the proof of Lemma 6.2.

REFERENCES

[1] K. Abrahamson, N. Dadoun, D. G. Kirkpatrick, and T. Przytycka, *A simple parallel tree contraction algorithm*, J. Algorithms, 10 (1989), pp. 287–302.

[2] N. Alon, L. Babai, and A. Itai, *A fast and simple randomized parallel algorithm for the maximal independent set problem*, J. Algorithms, 7 (1986), pp. 567–583.

[3] N. Alon, O. Goldreich, J. Håstad, and R. Peralta, *Simple constructions of almost k-wise independent random variables*, Random Structures Algorithms, 3 (1992), pp. 289–304.

[4] R. J. Anderson and G. L. Miller, *Deterministic parallel list ranking*, Algorithmica, 6 (1991), pp. 859–868.

[5] S. Arnborg, D. G. Corneil, and A. Proskurowski, *Complexity of finding embeddings in a k-tree*, SIAM J. Alg. Discrete Methods, 8 (1987), pp. 277–284.

[6] S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese, *An algebraic theory of graph reduction*, J. Assoc. Comput. Mach., 40 (1993), pp. 1134–1164.

[7] H. L. Bodlaender, NC-*algorithms for graphs with small treewidth*, in Proc. 14th International Workshop on Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science 344, J. van Leeuwen, ed., Springer-Verlag, Berlin, 1989, pp. 1–10.

[8] H. L. Bodlaender, *A linear time algorithm for finding tree-decompositions of small treewidth*, SIAM J. Comput., 25 (1996), pp. 1305–1317.

[9] H. L. Bodlaender, *On reduction algorithms for graphs with small treewidth*, in Proc. 19th International Workshop on Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science 790, J. van Leeuwen, ed., Springer-Verlag, Berlin, 1994, pp. 45–56.

[10] H. L. Bodlaender, *Improved self-reduction algorithms for graphs with bounded treewidth*, Discrete Appl. Math., 54 (1994), pp. 101–115.

[11] H. L. Bodlaender, *A Partial k-Arboretum of Graphs with Bounded Treewidth*, Tech. Report UU–CS–1996–02, Department of Computer Science, Utrecht University, Utrecht, the Netherlands, 1996; Theoret. Comput. Sci., to appear.

[12] H. L. Bodlaender and B. de Fluiter, *Parallel algorithms for series parallel graphs*, in Proc. 4th Annual European Symposium on Algorithms, Lecture Notes in Computer Science 1136, J. Diaz and M. Serna, eds., Springer-Verlag, Berlin, 1996, pp. 277–289.

[13] H. L. Bodlaender and T. Kloks, *Efficient and constructive algorithms for the pathwidth and treewidth of graphs*, J. Algorithms, 21 (1996), pp. 358–402.

[14] R. B. Borie, R. G. Parker, and C. A. Tovey, *Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families*, Algorithmica, 7 (1992), pp. 555–581.

[15] N. Chandrasekharan, *Fast Parallel Algorithms and Enumeration Techniques for Partial k-Trees*, Ph.D. thesis, Clemson University, Clemson, SC, 1989.

[16] N. Chandrasekharan and S. T. Hedetniemi, *Fast parallel algorithms for tree decomposing and parsing partial k-trees*, in Proc. 26th Annual Allerton Conference on Communication, Control, and Computing, Urbana-Champaign, IL, 1988, pp. 283–292.

[17] R. Cole and U. Vishkin, *Approximate parallel scheduling. Part* I: *The basic technique with applications to optimal parallel list ranking in logarithmic time*, SIAM J. Comput., 17 (1988), pp. 128–142.

[18] R. Cole and U. Vishkin, *The accelerated centroid decomposition technique for optimal parallel tree evaluation in logarithmic time*, Algorithmica, 3 (1988), pp. 329–346.

[19] R. Cole and U. Vishkin, *Faster optimal parallel prefix sums and list ranking*, Inform. and Comput., 81 (1989), pp. 334–352.

[20] B. Courcelle, *The monadic second-order logic of graphs* I. *Recognizable sets of finite graphs*, Inform. and Comput., 85 (1990), pp. 12–75.

[21] N. Dadoun and D. G. Kirkpatrick, *Parallel construction of subdivision hierarchies*, J. Comput. System Sci., 39 (1989), pp. 153–165.

[22] M. R. Fellows and M. A. Langston, *On search, decision, and the efficiency of polynomial-time algorithms*, J. Comput. System Sci., 49 (1994), pp. 769–779.

[23] B. de Fluiter and H. L. Bodlaender, *Parallel algorithms for treewidth two*, in Proc. 23rd International Workshop on Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science 1335, R. H. Möhring, ed., Springer-Verlag, Berlin, 1997, pp. 157–170.

[24] H. Gazit, G. L. Miller, and S.-H. Teng, *Optimal tree contraction in an* EREW *model*, in Concurrent Computations: Algorithms, Architecture, and Technology, S. K. Tewksbury, B. W. Dickinson, and S. C. Schwartz, eds., Plenum Press, New York, 1988, pp. 139–156.

[25] A. Gibbons and W. Rytter, *Optimal parallel algorithms for dynamic expression evaluation and context-free recognition*, Inform. and Comput., 81 (1989), pp. 32–45.

[26] A. V. Goldberg, S. A. Plotkin, and G. E. Shannon, *Parallel symmetry-breaking in sparse graphs*, SIAM J. Discrete Math., 1 (1988), pp. 434–446.

[27] D. Granot and D. Skorin–Kapov, NC *algorithms for recognizing partial* 2*-trees and* 3*-trees*, SIAM J. Discrete Math., 4 (1991), pp. 342–354.

[28] T. Hagerup, *Optimal parallel algorithms on planar graphs*, Inform. and Comput., 84 (1990), pp. 71–96.

[29] T. Hagerup, *Fast deterministic processor allocation*, J. Algorithms, 18 (1995), pp. 629–649.

[30] T. Hagerup, M. Chrobak, and K. Diks, *Optimal parallel* 5*-colouring of planar graphs*, SIAM J. Comput., 18 (1989), pp. 288–300.

[31] T. Hagerup, J. Katajainen, N. Nishimura, and P. Ragde, *Characterizations of k-terminal flow networks and computing network flows in partial k-trees*, in Proc. 6th Annual ACM-SIAM Symposium on Discrete Algorithms, 1995, pp. 641–649; J. Comput. System Sci., to appear.

[32] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers,* 5th ed., Oxford University Press, Oxford, 1979.

[33] S. Khuller and B. Schieber, *Efficient parallel algorithms for testing k-connectivity and finding disjoint s-t paths in graphs*, SIAM J. Comput., 20 (1991), pp. 352–375.

[34] S. R. Kosaraju and A. L. Delcher, *Optimal parallel evaluation of tree-structured computations by raking*, in Proc. 3rd Aegean Workshop on Computing, Lecture Notes in Computer Science 319, J. H. Reif, ed., Springer-Verlag, Berlin, 1988, pp. 101–110.

[35] J. Lagergren, *Efficient parallel algorithms for graphs of bounded tree-width*, J. Algorithms, 20 (1996), pp. 20–44.

[36] J. Lagergren and S. Arnborg, *Finding minimal forbidden minors using a finite congruence*, in Proc. 18th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science 510, J. Leach Albert, M. Rodríguez Artalejo, and B. Monien, eds., Springer-Verlag, Berlin, 1991, pp. 532–543.

[37] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and their Applications*, Cambridge University Press, Cambridge, 1986.

[38] J. Matoušek and R. Thomas, *Algorithms finding tree-decompositions of graphs*, J. Algorithms, 12 (1991), pp. 1–22.

[39] G. L. Miller and J. H. Reif, *Parallel tree contraction and its application*, in Proc. 26th Annual Symposium on Foundations of Computer Science, 1985, pp. 478–489.

[40] P. Ragde, *The parallel simplicity of compaction and chaining*, J. Algorithms, 14 (1993), pp. 371–380.

[41] S. Rajasekaran and J. H. Reif, *Optimal and sublogarithmic time randomized parallel sorting algorithms*, SIAM J. Comput., 18 (1989), pp. 594–607.

[42] B. A. Reed, *Finding approximate separators and computing tree width quickly*, in Proc. 24th Annual ACM Symposium on the Theory of Computing, 1992, pp. 221–228.

[43] N. Robertson and P. D. Seymour, *Graph minors.* XIII. *The disjoint paths problem*, J. Combin. Theory Ser. B., 63 (1995), pp. 65–110.

[44] R. E. Tarjan and U. Vishkin, *An efficient parallel biconnectivity algorithm*, SIAM J. Comput., 14 (1985), pp. 862–874.

[45] E. Wanke, *Bounded tree-width and* LOGCFL, J. Algorithms, 16 (1994), pp. 470–491.

SIAM J. COMPUT.
Vol. 27, No. 6, pp. 1747–1763, December 1998
© 1998 Society for Industrial and Applied Mathematics
011

FIRST-ORDER QUERIES ON FINITE STRUCTURES OVER THE REALS*

JAN PAREDAENS†, JAN VAN DEN BUSSCHE‡, AND DIRK VAN GUCHT§

Abstract. We investigate properties of finite relational structures over the reals expressed by first-order sentences whose predicates are the relations of the structure plus arbitrary polynomial inequalities, and whose quantifiers can range over the whole set of reals. In constraint programming terminology, this corresponds to Boolean real polynomial constraint queries on finite structures. The fact that quantifiers range over all reals seems crucial; however, we observe that each sentence in the first-order theory of the reals can be evaluated by letting each quantifier range over only a finite set of real numbers without changing its truth value. Inspired by this observation, we then show that when all polynomials used are linear, each query can be expressed uniformly on all finite structures by a sentence of which the quantifiers range only over the finite domain of the structure. In other words, linear constraint programming on finite structures can be reduced to ordinary query evaluation as usual in finite model theory and databases. Moreover, if only "generic" queries are taken into consideration, we show that this can be reduced even further by proving that such queries can be expressed by sentences using as polynomial inequalities only those of the simple form $x < y$.

Key words. first-order logic, linear arithmetic, relational databases, constraint programming

AMS subject classifications. Primary, 68P15; Secondary, 03C07, 03C10, 03C13

PII. S009753979629766

1. Introduction.

In this paper we are motivated by two fields of computer science which heavily rely on logic: relational databases and constraint programming. We will look at the latter from the perspective of the former.

In classical relational database theory [1], a database is modeled as a relational structure. The domain of this structure is some fixed universe **U** of possible data elements (such as all strings, or all natural numbers), and is typically infinite. The relations of the structure, in contrast, are always finite, as they model finite tables holding data. As a consequence, the *active* domain of the database, consisting of all data elements actually occurring in one or more of the relations, is finite as well.

A (Boolean) *query* is a mapping from databases (over some fixed relational signature) to true or false. A basic way of expressing a query is by a first-order sentence over the relational signature. For example, on a database containing information on children and hobbies, the query "does each parent have at least all hobbies of his children?" is expressed by the sentence $(\forall p)(\forall c)(\forall h)(\mathit{Child}(p,c) \wedge \mathit{Hobby}(c,h) \rightarrow \mathit{Hobby}(p,h))$.

Since the domain of each database is **U**, the quantifiers in a sentence expressing a query will naturally range over the whole infinite **U**. However, Aylamazyan et al. [5] showed that in order to obtain the result of the query it suffices to let the quantifiers range over the active domain augmented with a finite set of q additional data elements, where q is the number of quantified variables in the formula expressing the query. The

* Received by the editors February 5, 1996; accepted for publication (in revised form) October 9, 1996; published electronically June 3, 1998.
http://www.siam.org/journals/sicomp/27-6/29796.html
† Department of Mathematics and Computer Science, University of Antwerp (UIA), Universiteitsplein 1, B-2610 Antwerp, Belgium (pareda@uia.ac.be).
‡ Department WNI, University of Limburg (LUC), B-3590 Diepenbeek, Belgium (vdbuss@luc.ac.be).
§ Computer Science Department, Indiana University, Bloomington, IN 47405-4101 (vgucht@cs.indiana.edu).

intuition behind this result is that all data elements outside the active domain of a given database are alike with respect to that database.

Alternatively, we can choose to let the quantifiers range over the active domain only, thus obtaining a semantics which is quite different from the natural interpretation. For example, consider databases over the single unary relation symbol P. Then the sentence $(\forall x)P(x)$ will always be false under the natural interpretation, while under the active-domain interpretation it will always be true. In fact, it is not obvious that each query expressible under the natural interpretation is also expressible under the active-domain interpretation. Hull and Su [15] established that the implication indeed holds. (The converse implication holds as well, since the active-domain interpretation can easily be simulated under the natural interpretation using bounded quantification.)

In recent years, much attention has been paid to "constraint programming languages" (e.g., [9]). In particular, in 1990, Kanellakis, Kuper, and Revesz demonstrated that the idea of constraint programming also applies to database query languages by introducing the framework of "constraint query languages" [16]. An important instance of this framework is that of real polynomial constraints. Here, the universe $\mathbf{U}$ of data elements is the field $\mathbf{R}$ of real numbers. Databases, then, are relational structures over $\mathbf{R}$, but the database relations need no longer be finite; it suffices that they are definable as finite Boolean combinations of polynomial inequalities. In other words, each k-ary relation of the structure must be a semi-algebraic subset of $\mathbf{R}^k$ [10].

A basic way of querying real polynomial constraint databases is again by first-order sentences, which can now contain polynomial inequalities in addition to the predicate symbols of the relational signature. For example, if the database holds a set S of points in $\mathbf{R}^2$, the query "do all points in S lie on a common circle?" is expressed by $(\exists x_0)(\exists y_0)(\exists r)(\forall x)(\forall y)(S(x,y) \rightarrow (x-x_0)^2+(y-y_0)^2 = r^2)$. Note that quantifiers are naturally interpreted as ranging over the whole of $\mathbf{R}$. In order to evaluate such a sentence on a database, we replace each predicate symbol in the formula by the polynomial definition of the corresponding database relation, and obtain a sentence in the pure first-order theory of the reals. As is well known, this theory is decidable [22]; the truth value of the obtained sentence yields the result of the query. So, real polynomial constraint queries are effectively computable.

Finite relations are semi-algebraic, so that finite relational databases over the reals form an important special case of real polynomial constraint databases. For example, if we want to model a database holding a finite number of rectangles, we can either choose to store the full extents of the rectangles, resulting in the infinite set of all points on the rectangles (represented in terms of linear inequalities in the obvious way), or we can choose to store only the corner points of each rectangle, resulting in a finite relation.

In the present paper, we investigate whether the results by Aylamazyan et al. [5] and by Hull and Su [15], mentioned in the beginning of this Introduction, carry over from classical first-order queries on relational databases to polynomial constraint queries on finite databases over the reals. Indeed, as in the classical case, one can give an alternative active-domain semantics to constraint sentences and again ask whether this is without loss of expressive power. Note, however, that active-domain quantification defies the very nature of constraint programming as a means to reason about intentionally defined, potentially infinite, ranges of values. Hence, it is not obvious that the results just mentioned might carry over at all.

Nonetheless, we have found a natural analog of the Aylamazyan et al. theorem,

and we have been able to establish the verbatim analog of the Hull–Su theorem in the case when only linear polynomials are used. This is often the case in practice. Our result might be paraphrased by saying that on finite structures, first-order linear constraint programming can be reduced to ordinary query evaluation as usual in finite model theory and databases.

Our development is based upon the following observation. Consider a prenex normal form sentence $(Q_1x_1)\dots(Q_nx_n)M(x_1,\dots,x_n)$ in the first-order theory of the reals. For any finite set D_0 of real numbers, there exists a sequence $D_0 \subseteq D_1 \subseteq \cdots \subseteq D_n$ of finite sets of reals such that the sentence can be evaluated by letting each quantifier Q_i range over D_i only (rather than over the whole of $\mathbf{R}$) without changing the sentence's truth value. By taking D_0 to be the active domain of a given finite database over the reals, we get the analog in the real case of the Aylamazyan et al. theorem.

The reader familiar with Collins's method for quantifier elimination in real-closed fields through cylindrical algebraic decomposition (cad) [3, 4, 12] will not be surprised by the above observation. Indeed, it follows more or less directly from an obvious adaptation of the cad construction. However, we give an alternative, self-contained proof from first principles which abstracts away the purely algorithmical aspects of the cad construction and focuses on the logic behind it. Importantly, this proof provides us with a basis to show how, in the case of linear polynomials, the construction of the sequence $D_1 \subseteq \cdots \subseteq D_n$ departing from the active domain D_0 can be simulated using a linear constraint formula. As a result, we obtain the analog in the real case of the Hull–Su theorem.

In a final section of this paper, we look at queries that are "generic," i.e., that do not distinguish between isomorphic databases. Genericity is a natural criterion in the context of classical relational databases [2, 11]. Perhaps this is a little less so for databases over the reals; in other work [19] we have proposed alternative, "spatial" genericity criterions based on geometrical intuitions. Nevertheless, it remains interesting to investigate which classically generic queries can be expressed using linear constraint sentences.

Sentences that do not contain any polynomial inequalities always express generic queries, but from the moment a sentence even contains only simple inequalities of the form $x < y$ it can already be nongeneric. Furthermore, examples are known (e.g., [1, Exercise 17.27]) of generic queries expressible with such simple inequalities but not without. In other words, simple inequalities, though inherently nongeneric when viewed in isolation, help to express more generic queries. The natural question now is whether general linear polynomial inequalities help even more. We will answer this question negatively.[1]

This paper is organized as follows. We start with a rather general section 2 in which we introduce the notion of domain sequence on which much of our development will hinge. In section 3 we then introduce the subject of queries on real databases. In section 4 we focus on the linear case. In section 5 we discuss generic queries.

Since we presented the original ideas contained in the present paper at a conference [20], several researchers have been able to generalize our results. We provide a brief summary of these generalizations in section 6.

[1] We thus provide a partial rectification of Kuper's original intuitions [17] (which are incorrect as stated).

2. Domain sequences. We will use the basic terminology from mathematical logic [13]. Let $\mathcal{A}$ be a structure over a finite relational vocabulary L. The domain of $\mathcal{A}$ is denoted by A. Let $\Phi(x_1, \ldots, x_k)$ be a first-order formula over L written in prenex normal form

$$(*) \qquad (Q_{k+1}x_{k+1}) \ldots (Q_n x_n) M(x_1, \ldots, x_n),$$

with each Q_i either $\exists$ or $\forall$ and M quantifier-free. If $k = 0$ then Φ is a sentence; if $k = n$ then Φ is quantifier-free. If $\bar{a} = a_1, \ldots, a_k \in A$ is a tuple of elements in A then the truth of Φ in $\mathcal{A}$ with a_i substituted for x_i is denoted by $\mathcal{A} \models \Phi[\bar{a}]$.

If $D_{k+1}, \ldots, D_n$ are subsets of A, then we write

$$(\mathcal{A}; D_{k+1}, \ldots, D_n) \models \Phi[\bar{a}]$$

if $\Phi[\bar{a}]$ evaluates to true in $\mathcal{A}$ when we let each quantifier Q_i range over D_i only rather than over the whole of A.

Example 2.1. Let $\mathcal{A}$ consist of the integers together with the predicate $y = x^2$, and let Φ be the sentence $(\forall x)(\exists y) y = x^2$. Then $(\mathcal{A}; \{-1, 2\}, \{1, 4\}) \models \Phi$, but $(\mathcal{A}; \{-1, 2\}, \{1, 3\}) \not\models \Phi$.

In this section, we prove the following theorem.

THEOREM 2.2. *Let Φ be a sentence $(Q_1 x_1) \ldots (Q_n x_n) M(x_1, \ldots, x_n)$, and let D_0 be a finite subset of A. Then there exists an increasing sequence $D_0 \subseteq D_1 \subseteq \cdots \subseteq D_n$ of finite subsets of A such that*

$$\mathcal{A} \models \Phi \quad \Longleftrightarrow \quad (\mathcal{A}; D_1, \ldots, D_n) \models \Phi.$$

Example 2.3. As a trivial illustration, let $\mathcal{A}$ consist of the integers together with the predicate $x = y^2$, and let Φ be the sentence $(\forall x_1)(\exists x_2) x_2 = (x_1)^2$. Let D_0 be the empty set. We have $\mathcal{A} \models \Phi$, and indeed, for $D_1 = \{-1, 2\}$ and $D_2 = \{-1, 2, 1, 4\}$, we have $(\mathcal{A}; D_1, D_2) \models \Phi$.

To prove Theorem 2.2 we introduce various auxiliary notions on which we will also rely in later sections.

We will use the following natural equivalence relation on A^n.

DEFINITION 2.4. *Two points $\bar{a}$ and $\bar{b}$ in A^n are called equivalent, denoted $\bar{a} \equiv \bar{b}$, if for each atomic formula $F(x_1, \ldots, x_n)$ we have $\mathcal{A} \models F[\bar{a}]$ iff $\mathcal{A} \models F[\bar{b}]$. In model-theoretic terminology, $\bar{a}$ and $\bar{b}$ are equivalent if they are of the same basic type in $\mathcal{A}$.*

Example 2.5. Let $\mathcal{A}$ consist of the reals together with the predicates $C(x, y)\ \theta\ 0$, $L_1(x, y)\ \theta\ 0$, and $L_2(x, y)\ \theta\ 0$, where θ is $<$, $=$, or $>$, and C, L_1, and L_2 are polynomials describing the circle and two lines depicted in Figure 1. The same figure shows that there are 19 equivalence classes in A^2: $\{a\}$, $\{b, c\}$, $\{d, e\}$, A, $B \cup D$, C, E, $F \cup H$, G, I, $J \cup L$, K, α, β, γ, $\delta \cup \lambda$, ϵ, η, and κ.

We now extend this equivalence relation inductively to lower dimensions such that the equivalence classes at each dimension are "cylindrical" over the equivalence classes at the next lower dimension.

DEFINITION 2.6. *Let $i < n$ and assume $\equiv$ is already defined on A^{i+1}. Then for $\bar{a}, \bar{b} \in A^i$ we say $\bar{a} \equiv \bar{b}$ if for each $a_{i+1} \in A$ there is a $b_{i+1} \in A$ such that $(\bar{a}, a_{i+1}) \equiv (\bar{b}, b_{i+1})$ and conversely, for each b_{i+1} there is an a_{i+1} such that $(\bar{b}, b_{i+1}) \equiv (\bar{a}, a_{i+1})$.*

Example 2.7. In Figure 2 there are 12 equivalence classes in A: $\{p\}$, $\{q\}$, $\{r\}$, $\{s\}$, $\{t\}$, $\{u\}$, P, $Q \cup V$, R, S, T, and U.

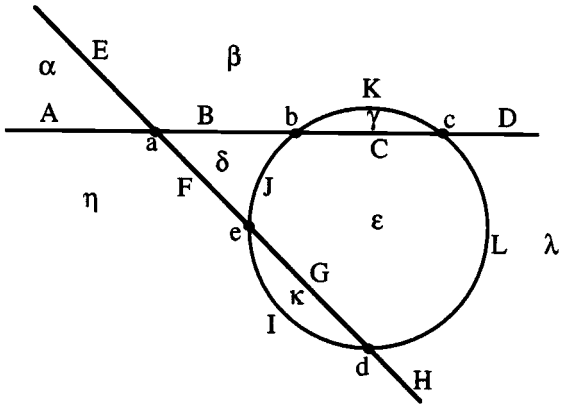


FIG. 1. *Equivalence classes in the plane induced by a circle and two lines.*

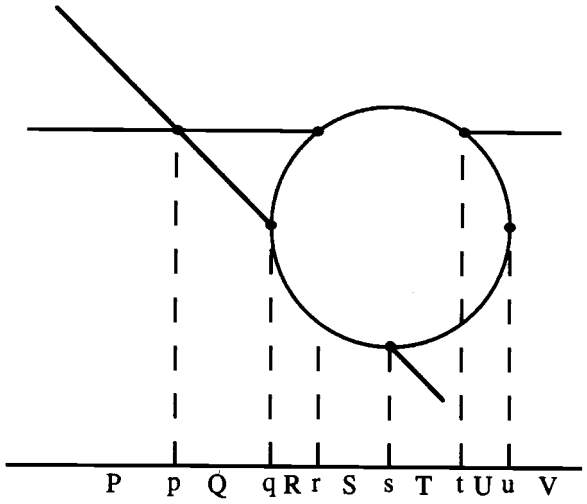


FIG. 2. *From equivalence classes in A^2 (Example* 2.5*) to equivalence classes in A.*

We note the following lemma for further use.

LEMMA 2.8. *For each i, $\equiv$ is of finite index on A^i.*

Proof. The proof is by downward induction on i. The base case $i = n$ is trivial since the number of atomic formulas $F(x_1, \ldots, x_n)$ is finite (we assumed a finite relational vocabulary). So assume $i < n$. For $\bar{a} \in A^i$, let $\kappa(\bar{a})$ be the set of equivalence classes in A^{i+1} intersecting the "vertical line through $\bar{a}$" $\{(\bar{a}, a_{i+1}) \mid a_{i+1} \in A\}$. Clearly, for $\bar{a}, \bar{b} \in A^i$, $\bar{a} \not\equiv \bar{b}$ implies $\kappa(\bar{a}) \neq \kappa(\bar{b})$. Since, by induction, $\equiv$ is of finite index on A^{i+1}, κ can have only a finite number of possible values and hence $\equiv$ is of finite index on A^i as well. □

The relevance of the equivalence relations just defined is demonstrated by the following lemma. We use the following notation: let $\Phi(\bar{x})$ be as in $(*)$ above. For $k \leq i \leq n$, $\Phi|_i$ stands for the formula

$$(Q_{i+1}x_{i+1}) \ldots (Q_n x_n) M(x_1, \ldots, x_n).$$

So, $\Phi|_k$ equals Φ and $\Phi|_n$ equals M.

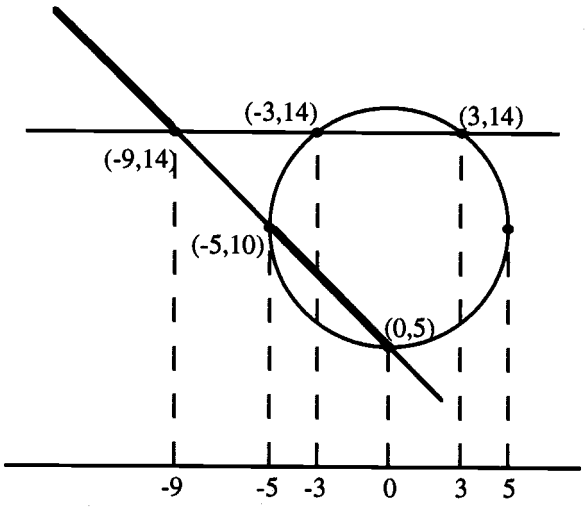


FIG. 3. *Equivalent points and formula satisfaction.*

LEMMA 2.9. *Let $k \leq i \leq n$, and let $\bar{a} \equiv \bar{b}$ be equivalent points in A^i. Then*

$$\mathcal{A} \models \Phi|_i[\bar{a}] \iff \mathcal{A} \models \Phi|_i[\bar{b}].$$

Proof. The proof is a straightforward downward induction on i. The base case, $i = n$ and $\Phi|_n$ being quantifier-free, is obvious. Now let $k \leq i < n$. We have $\Phi|_i = (Q_{i+1}x_{i+1})\Phi|_{i+1}$. We first consider the case $Q_{i+1} = \exists$. Note that we only have to prove the implication from left to right; the other direction follows by symmetry. If $\mathcal{A} \models \Phi|_i[\bar{a}]$ then there exists $a_{i+1} \in A$ such that $\mathcal{A} \models \Phi|_{i+1}[\bar{a}, a_{i+1}]$. Since $\bar{a} \equiv \bar{b}$, there exists $b_{i+1} \in A$ such that $(\bar{a}, a_{i+1}) \equiv (\bar{b}, b_{i+1})$. By the induction hypothesis, it follows that $\mathcal{A} \models \Phi|_{i+1}[\bar{b}, b_{i+1}]$ and hence $\mathcal{A} \models \Phi|_i[\bar{b}]$.

The case $Q_{i+1} = \forall$ is similar. If $\mathcal{A} \models \Phi|_i[\bar{a}]$ then for each $a_{i+1} \in A$ we have $\mathcal{A} \models \Phi|_{i+1}[\bar{a}, a_{i+1}]$. Since $\bar{a} \equiv \bar{b}$, for each $b_{i+1} \in A$ there exists an $a_{i+1} \in A$ such that $[\bar{b}, b_{i+1}] \equiv [\bar{a}, a_{i+1}]$. By the induction hypothesis, it follows that for each b_{i+1}, $\mathcal{A} \models \Phi|_{i+1}[\bar{b}, b_{i+1}]$ and hence $\mathcal{A} \models \Phi|_i[\bar{b}]$. □

Example 2.10. Continuing Examples 2.5 and 2.7, let $C(x, y) = x^2 + y^2 - 20x + 75$, $L_1(x, y) = x + y - 5$, and $L_2(x, y) = y - 14$. Let Φ be the sentence $(\forall x_1)(\exists x_2)(x_1 + x_2 - 5 = 0 \wedge (x_2 - 14 > 0 \vee (x_1)^2 + (x_2)^2 - 20x_1 + 75 < 0))$. This is illustrated in Figure 3. We have $\Phi|_0 = \Phi$, $\Phi|_1 = (\exists x_2)(x_1 + x_2 - 5 = 0 \wedge (x_2 - 14 > 0 \vee (x_1)^2 + (x_2)^2 - 20x_1 + 75 < 0))$, and $\Phi|_2 = x_1 + x_2 - 5 = 0 \wedge (x_2 - 14 > 0 \vee (x_1)^2 + (x_2)^2 - 20x_1 + 75 < 0)$. As can be deduced from Example 2.7 the equivalence classes in A are $(-\infty, -9)$, $[-9]$, $(-9, -5) \cup (5, \infty)$, $[-5]$, $(-5, -3)$, $[-3]$, $(-3, 0)$, $[0]$, $(0, 3)$, $[3]$, $(3, 5)$, and $[5]$. We have $\mathcal{A} \models \Phi|_1[a]$ for each $a \in (-\infty, -9) \cup (-5, -3) \cup [-3] \cup (-3, 0]$.

The notion of *domain sequence* is defined next.

DEFINITION 2.11. *A sequence $D_k \subseteq D_{k+1} \subseteq \cdots \subseteq D_n$ of finite subsets of A is called a* domain sequence *if for each $k \leq i < n$:*

$$\forall \bar{a} \in (D_i)^i,\ \forall a_{i+1} \in A,\ \exists a'_{i+1} \in D_{i+1} : (\bar{a}, a_{i+1}) \equiv (\bar{a}, a'_{i+1}).$$

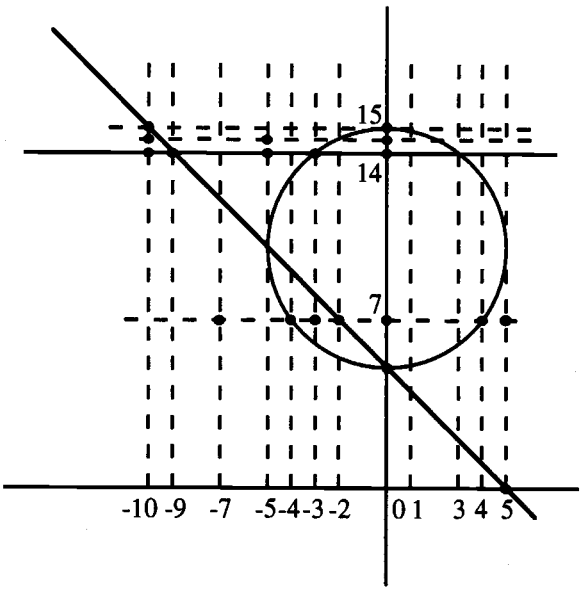


FIG. 4. *Domain sequence construction.*

Example 2.12. Continuing Example 2.10, from Figure 4 we see that (D_0, D_1, D_2), with

$D_0 = \{0\}$,

$D_1 = \{-10, -9, -7, -5, -4, -3, -2, 0, 1, 3, 4, 5\}$, and

$D_2 = D_1 \cup \{7, 14, 14.5, 15\}$,

is a domain sequence.

Since $\equiv$ is of finite index (Lemma 2.8), we know the following.

LEMMA 2.13. *For any given finite $D_k \subseteq A$, there exists a domain sequence starting from D_k.*

The following technical lemma now directly implies Theorem 2.2.

LEMMA 2.14. *Let*

- $D_k \subseteq D_{k+1} \subseteq \cdots \subseteq D_n$ *be a domain sequence*;
- $a_1, \ldots, a_k \in D_k$;
- $k \leq i \leq n$; *and*
- $a_j \in D_j$ *for* $k < j \leq i$.

Then

$$\mathcal{A} \models \Phi|_i[a_1, \ldots, a_i] \quad \Longleftrightarrow \quad (\mathcal{A}; D_{i+1}, \ldots, D_n) \models \Phi|_i[a_1, \ldots, a_i].$$

Proof. The proof is by downward induction on i. Denote $(a_1, \ldots, a_i)$ by $\bar{a}$. The case $i = n$ is trivial. So assume $i < n$. We have $\Phi|_i = (Q_{i+1}x_{i+1})\Phi|_{i+1}$. Consider first the case $Q_{i+1} = \exists$. For the implication from left to right, assume $\mathcal{A} \models \Phi|_i[\bar{a}]$. Then there exists $a_{i+1} \in A$ such that $\mathcal{A} \models \Phi|_{i+1}[\bar{a}, a_{i+1}]$. According to Definition 2.11, there exists $a'_{i+1} \in D_{i+1}$ such that $(\bar{a}, a_{i+1}) \equiv (\bar{a}, a'_{i+1})$. By Lemma 2.9, we also have $\mathcal{A} \models \Phi|_{i+1}[\bar{a}, a'_{i+1}]$. By induction, $(\mathcal{A}; D_{i+2}, \ldots, D_n) \models \Phi|_{i+1}[\bar{a}, a'_{i+1}]$. We can thus conclude that $(\mathcal{A}; D_{i+1}, \ldots, D_n) \models \Phi|_i[\bar{a}]$.

For the implication from right to left, assume $(\mathcal{A}; D_{i+1}, \ldots, D_n) \models \Phi|_i[\bar{a}]$. Then there exists $a_{i+1} \in D_{i+1}$ such that $(D_{i+2}, \ldots, D_n) \models \Phi|_{i+1}[\bar{a}, a_{i+1}]$. By induction, we have $\mathcal{A} \models \Phi_{i+1}[\bar{a}, a_{i+1}]$. Since a_{i+1} is trivially in A, we can thus conclude that $\mathcal{A} \models \Phi|_i[\bar{a}]$.

Next consider the case $Q_{i+1} = \forall$. For the implication from left to right, assume $\mathcal{A} \models \Phi|_i[\bar{a}]$. Then for each $a_{i+1} \in A$ we have $\mathcal{A} \models \Phi|_{i+1}[\bar{a}, a_{i+1}]$. In particular, this holds for each $a_{i+1} \in D_{i+1}$, and by induction, we have $(\mathcal{A}; D_{i+2}, \ldots, D_n) \models \Phi|_{i+1}[\bar{a}, a_{i+1}]$. We can thus conclude that $(\mathcal{A}; D_{i+1}, \ldots, D_n) \models \Phi|_i[\bar{a}]$.

For the implication from right to left, assume $(\mathcal{A}; D_{i+1}, \ldots, D_n) \models \Phi|_i[\bar{a}]$. Then for each $\alpha \in D_{i+1}$ we have $(\mathcal{A}; D_{i+2}, \ldots, D_n) \models \Phi|_{i+1}[\bar{a}, \alpha]$, and thus, by induction, also $\mathcal{A} \models \Phi|_{i+1}[\bar{a}, \alpha]$. Now take an arbitrary $a_{i+1} \in A$. According to Definition 2.11, there exists $a'_{i+1} \in D_{i+1}$ such that $(\bar{a}, a_{i+1}) \equiv (\bar{a}, a'_{i+1})$. By Lemma 2.9, since $\mathcal{A} \models \Phi|_{i+1}[\bar{a}, a'_{i+1}]$, we also have $\mathcal{A} \models \Phi|_{i+1}[\bar{a}, a_{i+1}]$. We can thus conclude that $\mathcal{A} \models \Phi_i[\bar{a}]$. ∎

COROLLARY 2.15. *Let Φ be a sentence $(Q_1 x_1) \ldots (Q_n x_n) M(x_1, \ldots, x_n)$, and let $D_0 \subseteq D_1 \subseteq \cdots \subseteq D_n$ be a domain sequence. Then*

$$\mathcal{A} \models \Phi \quad \Longleftrightarrow \quad (\mathcal{A}; D_1, \ldots, D_n) \models \Phi.$$

Proof. Set $i = k = 0$ in Lemma 2.14. ∎

3. Queries on real databases. Fix a relational vocabulary σ consisting of a finite number of relation symbols S with associated arity. A *real database* $\mathcal{B}$ is a structure of type σ having the set $\mathbf{R}$ of real numbers as domain, assigning to each relation symbol S of arity a in σ a *finite* relation $S^{\mathcal{B}}$ of rank a on $\mathbf{R}$.[2] The *active domain* of $\mathcal{B}$, denoted by $\mathrm{adom}(\mathcal{B})$, is the (finite) set of all real numbers appearing in one or more relations in $\mathcal{B}$.

A *query* is a mapping from databases of type σ to true or false. A basic way of expressing queries is by *query formulas*, which are standard first-order formulas built using Boolean connectives and quantification from atomic formulas of one of the following two forms:

- $p > 0$, with p a multivariate polynomial with real coefficients;
- $S(p_1, \ldots, p_a)$, with S a relation symbol in σ of arity a, and each p_i a polynomial as in the previous item.

If $\Phi(\bar{x})$ is a query formula and $\mathcal{B}$ is a database, then the truth of Φ in $\mathcal{B}$, denoted by $\mathcal{B} \models \Phi[\bar{a}]$, is defined in the standard way. In particular, if Φ is a sentence, it expresses the query yielding true on an input database $\mathcal{B}$ iff $\mathcal{B} \models \Phi$.

Example 3.1. Assume $\sigma = \{S\}$ with $\alpha(S) = 2$. The query "do all points in S lie on a common circle?" can be expressed as

$$(\exists x_0)(\exists y_0)(\exists r)(\forall x)(\forall y)(S(x, y) \rightarrow (x - x_0)^2 + (y - y_0)^2 = r^2).$$

(Conditions of the form $p = 0$ are expressible in terms of conditions of the form $p > 0$ as $\neg(p > 0) \wedge \neg(-p > 0)$.)

The query "is there a point in S whose coordinates are greater than or equal to 1?" can be expressed as $(\exists x)(\exists y) S(x^2 + 1, y^2 + 1)$. Note that the quantifiers are naturally interpreted as ranging over the whole of $\mathbf{R}$.

Formulas that do not mention any of the relation names in σ are called *real formulas*. Let Ψ be a real formula and let all variables occurring in Ψ be among

[2] Formally, $S^{\mathcal{B}} \subseteq \mathbf{R} \times \cdots \times \mathbf{R}$ (a times).

$x_1, \ldots, x_n$. Let Π be the set of all polynomials p for which the inequality $p > 0$ occurs in Ψ. For such a set Π of polynomials over the variables $x_1, \ldots, x_n$ we have the following definition.

DEFINITION 3.2. *The structure* $\mathbf{R}_\Pi$ *is the structure having as domain the set* $\mathbf{R}$ *of real numbers, and having as relations the* n*-ary relations* $\{(r_1, \ldots, r_n) \mid p(r_1, \ldots, r_n) > 0\}$ *for each* $p \in \Pi$. *If* Π *comes from a formula* Ψ*, as above, we will also refer to* $\mathbf{R}_\Pi$ *as* $\mathbf{R}_\Psi$. *Note that* Ψ *can be naturally evaluated in the structure* $\mathbf{R}_\Psi$.

If Φ is a query sentence and $\mathcal{B}$ is a database, then we can produce a real sentence $\Phi^{\mathcal{B}}$ in a very natural way as follows. Let $S(p_1, \ldots, p_a)$ be an atomic subformula of Φ, with S a relation symbol in σ. We know that $S^{\mathcal{B}}$ is a finite relation consisting of, say, the m tuples $\{(e_{11}, \ldots, e_{1a}), \ldots, (e_{m1}, \ldots, e_{ma})\}$. Then replace $S(p_1, \ldots, p_a)$ in Φ by $\bigvee_{i=1}^{m} p_1 = e_{i1} \wedge \ldots \wedge p_a = e_{ia}$. It is obvious that

$$\mathcal{B} \models \Phi \quad \Longleftrightarrow \quad \mathbf{R}_{\Phi^{\mathcal{B}}} \models \Phi^{\mathcal{B}}.$$

Now assume the query sentence Φ is in prenex normal form:

$$(Q_1 x_1) \ldots (Q_n x_n) M(x_1, \ldots, x_n). \tag{$\dagger$}$$

If $\mathcal{B}$ is a database and $D_1, \ldots, D_n$ are subsets of $\mathbf{R}$, then we say that Φ is satisfied on $(\mathcal{B}; D_1, \ldots, D_n)$, written $(\mathcal{B}; D_1, \ldots, D_n) \models \Phi$, if Φ evaluates to true on $\mathcal{B}$ when we let each quantifier Q_i range over D_i only, rather than over the whole of $\mathbf{R}$.

Corollary 2.15 immediately implies the following.

THEOREM 3.3. *Let* Φ *be a query sentence as in* ($\dagger$) *above and let* $\mathcal{B}$ *be a real database. For each domain sequence* $D_0 \subseteq D_1 \subseteq \cdots \subseteq D_n$ *in the context of the structure* $\mathbf{R}_{\Phi^{\mathcal{B}}}$,

$$\mathcal{B} \models \Phi \quad \Longleftrightarrow \quad (\mathcal{B}; D_1, \ldots, D_n) \models \Phi.$$

When we choose $D_0 = \mathrm{adom}(\mathcal{B})$, this theorem can be viewed as the analog in the real case of the Aylamazyan et al. theorem [5] mentioned in the Introduction.

4. The linear case. In this section, we focus on *linear* queries, expressed by query sentences in which all occurring polynomials are linear. We prove that each linear query is expressible by a linear query sentence wherein the quantifiers range over the active domain of the input database only. Thereto, we introduce a particular way to construct a domain sequence starting with the active domain of a database, based on Gaussian elimination. We then show that this construction can be simulated in a uniform (i.e., database-independent) way by a linear query formula.

Let Π be a set of linear polynomials on the variables $x_1, \ldots, x_n$. Recall Definition 3.2 of the structure $\mathbf{R}_\Pi$. Within the context of this structure we can consider equivalence of points in $\mathbf{R}^i$, for $i \leq n$, as defined in Definitions 2.4 and 2.6.

Each polynomial $p \in \Pi$ is of the form $c_0^p + \sum_{j=1}^{n} c_j^p x_j$. We define a sequence $\Pi_n, \ldots, \Pi_1$ of linear polynomials inductively as follows.

DEFINITION 4.1. $\Pi_n = \Pi$, *and for* $i < n$,

$$\Pi_i = \{p \in \Pi_{i+1} \mid c_{i+1}^p = 0\} \cup \{p \cdot c_{i+1}^q - q \cdot c_{i+1}^p \mid p, q \in \Pi_{i+1},\ c_{i+1}^p \neq 0 \neq c_{i+1}^q\}.$$

In other words, each Π_i *is a set of linear polynomials over* $x_1, \ldots, x_i$ *obtained from* Π_{i+1} *by Gaussian elimination.*

In the next proposition, equivalence of points in $\mathbf{R}^i$ with respect to $\mathbf{R}_\Pi$ will be characterized in terms of the polynomials in Π_i. Thereto we need an easy-to-prove lemma.

LEMMA 4.2. *Let α_1, α_2, β_1, and β_2 be elements from some densely ordered domain. The following are equivalent.*

1. *For $(i,j) \in \{(1,2),(2,1)\}$,*
$$\alpha_i > \alpha_j \quad \Leftrightarrow \quad \beta_i > \beta_j.$$
2. *For each α there exists β such that for $i = 1, 2$,*
$$\alpha_i > \alpha \quad \Leftrightarrow \quad \beta_i > \beta,$$
and conversely, for each β there exists α such that the same holds.

PROPOSITION 4.3. *Let $1 \leq i \leq n$ and let $\bar{a}, \bar{b} \in \mathbf{R}^i$. Then $\bar{a}$ and $\bar{b}$ are equivalent with respect to $\mathbf{R}_\Pi$ if and only if for each polynomial p in Π_i,*
$$p(\bar{a}) > 0 \quad \Leftrightarrow \quad p(\bar{b}) > 0.$$

Proof. The proof is by downward induction on i. The case $i = n$ is just the definition of equivalence of n-tuples. So assume $i < n$. Let $\bar{a} \equiv \bar{b}$. Then for each a_{i+1} there is a b_{i+1} such that $(\bar{a}, a_{i+1}) \equiv (\bar{b}, b_{i+1})$ (and conversely). Equivalently, by induction, for each a_{i+1} there is a b_{i+1} such that for each polynomial p in Π_{i+1}, $p(\bar{a}, a_{i+1}) > 0 \Leftrightarrow p(\bar{b}, b_{i+1}) > 0$. If $c^p_{i+1} = 0$ then $p \in \Pi_i$ and we get $p(\bar{a}) > 0 \Leftrightarrow p(\bar{b}) > 0$; this deals with the first kind of elements of Π_i.

For the other kind of elements of Π_i, consider $p, q \in \Pi_{i+1}$ with $c^p_{i+1} \neq 0 \neq c^q_{i+1}$. From the above, for each a_{i+1} there is a b_{i+1} such that

$$\left(c^p_0 + \sum_{j=1}^{i} c^p_j a_j\right) / c^p_{i+1} > -a_{i+1} \quad \Leftrightarrow \quad \left(c^p_0 + \sum_{j=1}^{i} c^p_j b_j\right) / c^p_{i+1} > -b_{i+1}$$

and

$$\left(c^q_0 + \sum_{j=1}^{i} c^q_j a_j\right) / c^q_{i+1} > -a_{i+1} \quad \Leftrightarrow \quad \left(c^q_0 + \sum_{j=1}^{i} c^q_j b_j\right) / c^q_{i+1} > -b_{i+1}.$$

Conversely, for each b_{i+1} there is an a_{i+1} such that the same holds. By Lemma 4.2 we thus deduce

$$\left(c^p_0 + \sum_{j=1}^{i} c^p_j a_j\right) / c^p_{i+1} > \left(c^q_0 + \sum_{j=1}^{i} c^q_j a_j\right) / c^q_{i+1}$$
$$\Leftrightarrow \quad \left(c^p_0 + \sum_{j=1}^{i} c^p_j b_j\right) / c^p_{i+1} > \left(c^q_0 + \sum_{j=1}^{i} c^q_j b_j\right) / c^q_{i+1}$$

and hence

$$\left(c^p_0 + \sum_{j=1}^{i} c^p_j a_j\right) \cdot c^q_{i+1} > \left(c^q_0 + \sum_{j=1}^{i} c^q_j a_j\right) \cdot c^p_{i+1}$$
$$\Leftrightarrow \quad \left(c^p_0 + \sum_{j=1}^{i} c^p_j b_j\right) \cdot c^q_{i+1} > \left(c^q_0 + \sum_{j=1}^{i} c^q_j b_j\right) \cdot c^p_{i+1}$$

or

$$(p \cdot c_{i+1}^q - q \cdot c_{i+1}^p)(\bar{a}) > 0 \quad \Leftrightarrow \quad (p \cdot c_{i+1}^q - q \cdot c_{i+1}^p)(\bar{b}) > 0,$$

which, by the definition of Π_i, is what had to be proven. This argument for the "only-if" implication can simply be reversed to prove the "if" implication. □

Now let Φ be a linear query sentence $(Q_1 x_1) \dots (Q_n x_n) M$ in prenex normal form, and let $\mathcal{B}$ be a database. Recall the definition of the real formula $\Phi^{\mathcal{B}}$ described in the previous section; note that since Φ is linear, $\Phi^{\mathcal{B}}$ is linear as well.

DEFINITION 4.4. *Fix Π to be the set of all polynomials occurring in $\Phi^{\mathcal{B}}$ plus all those of the form $p_i - e$, where p_i occurs in some atomic formula $S(p_1, \dots, p_a)$ of Φ and $e \in \mathrm{adom}(\mathcal{B})$. We can then consider the sequence $\Pi = \Pi_n, \dots, \Pi_1$ as in Definition 4.1. For what follows it is important to note that, since Π contains at least all polynomials occurring in $\Phi^{\mathcal{B}}$, equivalence of points with respect to $\mathbf{R}_\Pi$ implies equivalence with respect to the structure $\mathbf{R}_{\Phi^{\mathcal{B}}}$ mentioned in Theorem 3.3.*

Example 4.5. Let $\Phi = (\forall x_1)(\exists x_2)(S(x_1 + x_2 + 1) \wedge (x_1 + 2x_2 + 2 = 0))$. Then $\Pi = \Pi_2 = \{1 - e + x_1 + x_2 \mid e \in \mathrm{adom}(\mathcal{B})\} \cup \{2 + x_1 + 2x_2\}$ and $\Pi_1 = \{-2e + x_1 \mid e \in \mathrm{adom}(\mathcal{B})\}$.

We observe the following.

LEMMA 4.6. *Let $1 \le i \le n$. Then Π_i is a finite union of sets of the form*

$$\left\{ c_0 + \sum_{j=1}^{2^{(n-i)}} d_j e_j + \sum_{j=1}^{i} c_j x_j \mid e_1, \dots, e_{2^{(n-i)}} \in \mathrm{adom}(\mathcal{B}) \right\}.$$

Neither the number of these sets nor the coefficients c_i and d_i for each set depend on the particular database $\mathcal{B}$.

Proof. The proof is by downward induction on i. The base case $i = n$ is clear since $\Pi_n = \Pi$ is clearly of the good form. So assume $i < n$. By definition, Π_i is a union of two sets:

$$\{p \in \Pi_{i+1} \mid c_{i+1}^p = 0\}$$

and

$$\{p \cdot c_{i+1}^q - q \cdot c_{i+1}^p \mid p, q \in \Pi_{i+1},\ c_{i+1}^p \neq 0 \neq c_{i+1}^q\}.$$

By the induction hypothesis, the first set is clearly of the good form. Also, by the induction hypothesis, the second set is a finite union of sets of the form

$$\left\{ c'_{i+1} \left(c_0 + \sum_{j=1}^{2^{(n-i-1)}} d_j e_j + \sum_{j=1}^{i+1} c_j x_j \right) - c_{i+1} \left(c'_0 + \sum_{j=1}^{2^{(n-i-1)}} d'_j e'_j + \sum_{j=1}^{i+1} c'_j x_j \right) \mid \right.$$
$$\left. {}_1, \dots, e_{2^{(n-i-1)}}, e'_1, \dots, e'_{2^{(n-i-1)}} \in \mathrm{adom}(\mathcal{B}) \right\}.$$

After simplification, this is readily seen to be of the good form also. □

We are now in a position to define a particular domain sequence with respect to the structure $\mathbf{R}_{\Phi^{\mathcal{B}}}$, based on the sequence $\Pi_1, \dots, \Pi_n$. The sequence is inductively constructed: D_0 is empty and D_i $(i > 0)$ is constructed as follows. First consider the set E_i of all the ith coordinates of the i-dimensional points that are in a hyperplane

of Π_i and whose first $i-1$ coordinates are in D_{i-1}. Add D_{i-1} to E_i, resulting in D'_i. Finally, to be sure to obtain a point in every equivalence class, we add the mean value of every pair of elements of D'_i, as well as every element increased by one and every element decreased by one, resulting in D_i. Formally, we make the following definition.

DEFINITION 4.7. *The* linear sequence on $\mathcal{B}$ with respect to Φ *is the sequence* $\emptyset = D_0 \subseteq D_1 \subseteq \cdots \subseteq D_n$ *inductively defined as follows: for* $1 \leq i \leq n$, D_i *equals*

$$D'_i \cup \left\{ y \mid (\exists y_1, y_2) \in D'_i : y = \frac{y_1 + y_2}{2} \vee y = y_1 - 1 \vee y = y_1 + 1 \right\},$$

where D'_i *is* $D_{i-1} \cup E_i$ *with*

$$E_i = \left\{ -c_0/c_i - \sum_{j=1}^{2^{(n-i)}} (d_j/c_i)e_j - \sum_{j=1}^{i-1} (c_j/c_i)y_j \mid c_0 + \sum_{j=1}^{2^{(n-i)}} d_j e_j + \sum_{j=1}^{i} c_j x_j \in \Pi_i, \right.$$
$$\left. c_i \neq 0,\ y_1, \ldots, y_{i-1} \in D_{i-1},\ e_1, \ldots, e_{2^{(n-i)}} \in \mathrm{adom}(\mathcal{B}) \right\}.$$

Example 4.8. In Example 4.5 we have

$$D'_1 = \{2e_1 \mid e_1 \in \mathrm{adom}(\mathcal{B})\},$$

$$D_1 = \big\{2e_1 + \eta \mid \eta \in \{-1, 0, 1\}\big\},$$

and

$$E_2 = \big\{-1 + e_3 - (2e_1 + \eta),$$
$$-1 + e_3 - (e_1 + e_2),$$
$$-1 - (2e_1 + \eta)/2,$$
$$-1 - (e_1 + e_2)/2 \mid$$
$$e_1, e_2, e_3 \in \mathrm{adom}(\mathcal{B}),\ \eta \in \{-1, 0, 1\}\big\}.$$

PROPOSITION 4.9. *The linear sequence on* $\mathcal{B}$ *with respect to* Φ *is a domain sequence with respect to* $\mathbf{R}_{\Phi\mathcal{B}}$.

Proof. According to Definition 2.11, we must show for each $1 \leq i \leq n$ that

$$\forall \bar{a} \in (D_{i-1})^{i-1},\ \forall a_i \in \mathbf{R},\ \exists a'_i \in D_i : (\bar{a}, a_i) \equiv (\bar{a}, a'_i).$$

So, let $\bar{a} \in (D_{i-1})^{i-1}$ and assume $a_i \notin D_i$. Consider the definition of D_i in terms of $D'_i = D_{i-1} \cup E_i$ from Definition 4.7 above. We distinguish the following possibilities for a_i.

1. $a_i < \min(E_i)$; then put $a'_i := \min(E_i) - 1$.
2. $a_i > \max(E_i)$; then put $a'_i := \max(E_i) + 1$.
3. $\min(E_i) < a_i < \max(E_i)$; then put $a'_i := (e_1 + e_2)/2$, where e_1 is the maximal element in E_i such that $e_1 < a_i$, and e_2 is the minimal element such that $a_i < e_2$.

It is obvious that $a_i' \in D_i$; moreover, we invite readers to convince themselves that from the way E_i is defined, it follows that all polynomials in Π_i have the same sign on $(\bar{a}, a_i)$ and $(\bar{a}, a_i')$. Hence, by Proposition 4.3, the proposition follows. □

After one final lemma we will be able to state and prove the main result of this section.

LEMMA 4.10. *For each $0 \leq i \leq n$ there exists a finite set P of linear polynomials such that for each database $\mathcal{B}$, the ith member D_i of the linear sequence on $\mathcal{B}$ with respect to Φ equals $\{p(y_1, \ldots, y_z) \mid y_1, \ldots, y_z \in \mathrm{adom}(\mathcal{B}) \wedge p \in P\}$, with z independent of $\mathcal{B}$.*

Proof. The proof is by induction on i. The case $i = 0$ is trivial since $D_0 = \emptyset$ (put $P := \emptyset$). So assume $i > 0$. The definition of D_i in terms of D_i' in Definition 4.7 is clearly of the form $D_i = \{p(y_1, y_2) \mid y_1, y_2 \in D_i' \wedge p \in P'\}$ where P' consists of the four polynomials $(y_1 + y_2)/2$, $y_1 - 1$, $y_1 + 1$, and y_1. We have $D_i' = D_{i-1} \cup E_i$, where E_i is clearly of the form $\{p(y_1, \ldots, y_{i-1}, e_1, \ldots, e_{2^{(n-i)}}) \mid y_1, \ldots, y_{i-1} \in D_{i-1} \wedge e_1, \ldots, e_{2^{(n-i)}} \in \mathrm{adom}(\mathcal{B}) \wedge p \in P''\}$ for some P'', and by induction, D_{i-1} is of the form $\{p(y_1, \ldots, y_z) \mid y_1, \ldots, y_z \in \mathrm{adom}(\mathcal{B}) \wedge p \in P'''\}$ for some P'''. By combining these expressions using a tedious but straightforward substitution process, we obtain the desired form for D_i. □

THEOREM 4.11. *For each linear query sentence Φ there is a linear query sentence Ψ, which can be effectively constructed from Φ, such that for each database $\mathcal{B}$, $\mathcal{B} \models \Phi$ if and only if $\mathcal{B} \models_{\mathrm{adom}} \Psi$, where $\models_{\mathrm{adom}}$ denotes that the quantifiers in Ψ range over the active domain of the database only.*

Proof. Let $\emptyset \subseteq D_1 \subseteq \cdots \subseteq D_n$ be the linear sequence on $\mathcal{B}$ with respect to Φ. By Theorem 3.3 and Proposition 4.9, we know that $\mathcal{B} \models \Phi$ iff $(\mathcal{B}; D_1, \ldots, D_n) \models \Phi$. We can write the latter explicitly as $\mathcal{B} \models (Q_1 x_1 \in D_1) \ldots (Q_n x_n \in D_n) M(x_1, \ldots, x_n)$. From Lemma 4.10 we know that D_1 can be written as $\{p(y_1, \ldots, y_z) \mid y_1, \ldots, y_z \in \mathrm{adom}(\mathcal{B}) \wedge p \in P\}$. If Q_1 is $\exists$, we can rewrite the above formula as

$$\mathcal{B} \models (\exists y_1) \ldots (\exists y_z) \bigvee_{p \in P} (Q_2 x_2 \in D_2) \ldots (Q_n x_n \in D_n) M(p(y_1, \ldots, y_z), x_2, \ldots, x_n),$$

where each $(\exists y_i)$ ranges only over $\mathrm{adom}(\mathcal{B})$. If Q_1 is $\forall$ we have

$$\mathcal{B} \models (\forall y_1) \ldots (\forall y_z) \bigwedge_{p \in P} (Q_2 x_2 \in D_2) \ldots (Q_n x_n \in D_n) M(p(y_1, \ldots, y_z), x_2, \ldots, x_n).$$

By replacing $Q_2, \ldots, Q_n$ in a similar manner, we obtain the desired sentence Ψ.

If $\mathrm{adom}(\mathcal{B})$ is empty then the above strategy will not work. However, the sentence Ψ obtained above can be modified so as to test for this special case, and if this test succeeds, a fixed truth value can be returned. This fixed truth value is the result of evaluating $(\mathcal{B}_\emptyset \models \Phi)$, where $\mathcal{B}_\emptyset$ denotes the database with empty active domain.[3] □

5. Generic queries. Two databases $\mathcal{B}$ and $\mathcal{B}'$ over the same relational signature σ are called *isomorphic* if there is a bijection $\rho : \mathrm{adom}(\mathcal{B}) \to \mathrm{adom}(\mathcal{B}')$ such that $\rho(S^{\mathcal{B}}) = S^{\mathcal{B}'}$ for each relation symbol S in σ. A query which yields the same result on isomorphic databases is called *generic*.

[3] An exception occurs when the signature σ contains relation symbols of arity zero. In this case, there is no unique $\mathcal{B}_\emptyset$, but rather a fixed finite number of them. The sentence can test which one it is dealing with and return the appropriate truth value.

For example, assume that σ consists of a single binary relation symbol S. Databases of type σ can be viewed as finite directed graphs whose nodes are real numbers. Of course, any query expressed in the language $\mathcal{L}$ of pure first-order sentences over σ (i.e., not containing any polynomial inequalities) is generic. Other examples of generic queries are "is the graph connected?" or "is the number of edges even?".

In the language $\mathcal{L}^<$ consisting of those query sentences where all inequalities are of the simple form $x < y$ (with x and y variables),[4] nongeneric queries can easily be expressed, such as $(\forall x)(\forall y)S(x,y) \to x < y$. As pointed out in the Introduction, however, there are generic queries expressible in $\mathcal{L}^<$ but not in $\mathcal{L}$. We have been able to prove that there is no similar gain in expressiveness when moving from $\mathcal{L}^<$ to full linear query sentences.

THEOREM 5.1. *For each linear query sentence Φ expressing a generic query there is a query sentence Ψ in $\mathcal{L}^<$, which can be effectively constructed from Φ, such that for each database $\mathcal{B}$, $\mathcal{B} \models_{\text{adom}} \Phi$ if and only if $\mathcal{B} \models_{\text{adom}} \Psi$.*

As in Theorem 4.11, $\models_{\text{adom}}$ denotes that quantifiers range over the active domain only; we know by Theorem 4.11 that this active-domain interpretation is without loss of generality.

We next present an elementary proof of Theorem 5.1 based on three lemmas and one auxiliary definition.

The following fact is easy to prove.

LEMMA 5.2. *Let $q(x) = \sum_{i=0}^{d} a_i x^i$ be a polynomial with real coefficients, in one variable, of degree d ($a_d \neq 0$). Let*

$$r > d \cdot \frac{\max_{0 \leq i \leq d} |a_i|}{\min_{\substack{0 \leq j \leq d \\ a_j \neq 0}} |a_j|}.$$

Then $q(r)$ has the same sign as a_d.

We make the following definition.

DEFINITION 5.3. *Let $p(x_1, \ldots, x_n) = \sum_{i=1}^{n} b_i x_i + b_0$ be a linear polynomial with real coefficients in n variables. We associate with p a function $\Xi_p : \mathbf{R}^n \to \mathbf{R}$ as follows. Consider $\bar{y} = (y_1, \ldots, y_n) \in \mathbf{R}^n$. Associate a "weight" $W_p^{\bar{y}}(y_i)$ with each y_i by*

$$W_p^{\bar{y}}(y_i) := \sum_{\substack{1 \leq j \leq n \\ y_j = y_i}} b_j.$$

If all weights are zero, define $\Xi_p(y_1, \ldots, y_n) := b_0$. Otherwise, define

$$\Xi_p(y_1, \ldots, y_n) := W_p^{\bar{y}}(y_M),$$

where y_M is maximal with nonzero weight; i.e., $y_m = \max\{y_i \mid 1 \leq i \leq n,\ W_p^{\bar{y}}(y_i) \neq 0\}$.

Example 5.4. We illustrate the above definition with three examples.

1. Let $p(x_1, x_2, x_3) = x_1 - 5x_2 + 3x_3 - 8$. Then
   - $\Xi_p(7, 3, 9) = 3$;
   - $\Xi_p(7, 9, 3) = -5$.

[4] As an aside, we would like the reader to note that Theorem 4.11 specializes to query sentences in $\mathcal{L}^<$. This follows from general results in [8], but can also be proven in a direct way using an argument similar to our proof of Theorem 4.11.

2. Let $p(x_1, x_2) = 2x_1 - 2x_2 + 5$. Then $\Xi_p(5, 5) = 5$.
3. Let $p(x_1, \ldots, x_7) = 3x_1 - 3x_2 + 6x_3 - 6x_4 + 5x_5 + x_6 - 6x_7$. Then
   - $\Xi_p(4, 4, 2, 2, 2, 2, 2) = 0$;
   - $\Xi_p(5, 5, 4, 4, 3, 2, 1) = 5$;
   - $\Xi_p(5, 5, 4, 4, 1, 2, 3) = -6$.

The relevance of Ξ_p stems from the following observation.

LEMMA 5.5. *Let $p(x_1, \ldots, x_n) = \sum_{i=1}^{n} b_i x_i + b_0$ and let $s > 0$ be a natural number. Then there exists a number α_p such that for all $\beta > \alpha_p$ and for any sequence of integers $\bar{z} = z_1, \ldots, z_n \in \{0, \ldots, s\}$,*

$$p(\beta^{z_1}, \ldots, \beta^{z_n}) > 0 \quad \Longleftrightarrow \quad \Xi_p(z_1, \ldots, z_n) > 0.$$

Proof. Note that

$$p(\beta^{z_1}, \ldots, \beta^{z_n}) = \sum_{i=1}^{n} b_i \beta^{z_i} + b_0 = \sum_{j} W_p^{\bar{z}}(z_j) \beta^{z_j} + b_0,$$

where j in the latter sum ranges over a set of indices consisting of one j for each distinct value z_j. Hence, the value $p(\beta^{z_1}, \ldots, \beta^{z_n})$ can be viewed as the value of a univariate polynomial q in β. The highest-degree coefficient of q is $\Xi_p(z_1, \ldots, z_n)$. The degree of q is $\max_j z_j$. Hence, if we take

$$\alpha_p = s \cdot \frac{\max_B |B|}{\min_{|B| \neq 0} |B|},$$

where B ranges over all partial sums of b_i's, then any $\beta > \alpha_p$ satisfies the condition of Lemma 5.2 and thus, for such β, $p(\beta^{z_1}, \ldots, \beta^{z_n}) = q(\beta)$ has the same sign as $\Xi_p(z_1, \ldots, z_n)$. □

As a last lemma towards the proof of Theorem 5.1 we note the following.

LEMMA 5.6. *For a fixed polynomial p as above, the predicate $\Xi_p(y_1, \ldots, y_n) > 0$ can be expressed by a formula $\xi_p(y_1, \ldots, y_n)$ in $\mathcal{L}^<$.*

Proof. The crucial observation is that the value of $\Xi_p(y_1, \ldots, y_n)$ depends not on the actual values of the y_i's but only on their relative positions (in model-theoretic terms, the *order type* of $y_1, \ldots, y_n$). The number of possible order types (n being fixed) is finite, so the formula ξ_p simply consists of the disjunction of those order types for which the value $\Xi_p(y_1, \ldots, y_n)$ is positive. □

Example 5.7. Let $p(x_1, x_2) = 2x_1 - 2x_2 + 5$. Then $\Xi_p(y_1, y_2) > 0$ is expressed by $y_1 = y_2 \vee y_1 > y_2$.

Proof of Theorem 5.1. Replace in Φ every inequality $p(x_1, \ldots, x_n) > 0$ by the formula $\xi_p(x_1, \ldots, x_n) > 0$ of Lemma 5.6. In this way we obtain a query sentence Ψ in $\mathcal{L}^<$. We still have to show that $\mathcal{B} \models_{\text{adom}} \Phi$ iff $\mathcal{B} \models_{\text{adom}} \Psi$. Let s be the cardinality of $\text{adom}(\mathcal{B})$. Each polynomial p occurring in Φ has an associated lower bound α_p of Lemma 5.5. Let β be larger than any of these α_p's and let ρ be an order-preserving (i.e., monotone) bijection from $\text{adom}(\mathcal{B})$ to $\{\beta, \beta^2, \ldots, \beta^s\}$. Then

$$\begin{aligned} \mathcal{B} \models_{\text{adom}} \Phi &\Leftrightarrow \rho(\mathcal{B}) \models_{\text{adom}} \Phi \\ &\Leftrightarrow \rho(\mathcal{B}) \models_{\text{adom}} \Psi \\ &\Leftrightarrow \mathcal{B} \models_{\text{adom}} \Psi. \end{aligned}$$

The first equivalence holds since Φ is generic, the second holds by Lemma 5.5, and the third holds since ρ is monotone and Ψ is a formula in $\mathcal{L}^<$; the truth of formulas in $\mathcal{L}^<$ is preserved under order-preserving isomorphisms.

Inspection of the above proof shows that Theorem 5.1 can be sharpened a little bit. Indeed, of the given that Φ expresses a generic query, we actually use only that this query yields the same result on databases that are isomorphic via an order-preserving, rather than an arbitrary, bijection.

We can conclude that all generic queries that are not expressible in $\mathcal{L}^<$ are not expressible as a linear query either (by Theorem 4.11, under both the active-domain interpretation and the natural interpretation). In particular, this holds for the queries, already mentioned at the beginning of this section, of testing for connectivity or even cardinality of a finite graph over the reals.

COROLLARY 5.8. *Graph connectivity and even cardinality are not expressible by linear query sentences.*

Proof. By the above it suffices to show that these queries are not expressible in $\mathcal{L}^<$ with quantification on the active domain. But this is well known [1]. □

Grumbach, Su, and Tollu [14] have also obtained inexpressibility results for linear queries, using complexity arguments. In particular, they showed that in the context of the rationals $\mathbf{Q}$ rather than the reals $\mathbf{R}$, linear queries are in the complexity class AC_0, while even cardinality and connectivity are not. We would like to point out (as is readily verified) that our technical development applies equally well to the rationals.

6. Concluding remarks. Since we presented the original ideas contained in the present paper at a conference [20], several researchers have been able to generalize our results:

- In this paper we have considered databases and queries over the structure $\mathbf{R}$ of the reals. One can do the same for any arbitrary fixed infinite "universe-structure." Using the Ehrenfeucht–Mostowski theorem on first-order indiscernibles, Otto and Van den Bussche [18] have shown that Theorem 5.1 generalizes from $\mathbf{R}$ to any arbitrary fixed infinite structure.
- Benedikt et al. [7] have generalized Theorem 5.1 in two senses: again, to more universes than just the reals, and more importantly, to quantification on the whole universe rather than on the active domain only. One consequence of the results in [7] is a generalization of our Corollary 5.8: graph connectivity and even cardinality are not expressible by any (not necessarily linear) real query sentence, under both the natural interpretation and the active-domain interpretation.
- Belegradek, Stolboushkin, and Taitslin [21, 6] have generalized Theorem 5.1 in another sense: instead of finite databases they considered possibly infinite databases definable by real formulas involving only simple inequalities.
- Benedikt and Libkin [8] have shown that Theorem 4.11 holds in any densely ordered structure that satisfies the property of *o-minimality* and admits elimination of quantifiers. In particular, their result implies that Theorem 4.11 generalizes to the nonlinear case (since the structure of the reals with addition and multiplication admits elimination of quantifiers).

In constrast to the proofs of these generalizations, the proofs of our results as given in the present paper are elementary and constructive (the results in [21] are also constructive).

Acknowledgment. We are grateful to Bart Kuijpers for his careful reading of earlier drafts of the material presented in this paper, to Alex Stolboushkin for helpful

comments on a first presentation of our results, and to an anonymous referee for pointing out a mistake in the submitted draft of this paper.

REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*, Addison-Wesley, Reading, MA, 1994.

[2] A. Aho and J. Ullman, *Universality of data retrieval languages*, in Proc. ACM Symposium on Principles of Programming Languages, ACM Press, New York, 1979, pp. 110–120.

[3] D. Arnon, *Geometric reasoning with logic and algebra*, Artificial Intelligence, 37 (1988), pp. 37–60.

[4] D. Arnon, G. Collins, and S. McCallum, *Cylindrical algebraic decomposition,* I*: The basic algorithm*, SIAM J. Comput., 13 (1984), pp. 865–877.

[5] A. Aylamazyan, M. Gilula, A. Stolboushkin, and G. Schwartz, *Reducation of the relational model with infinite domains to the case of finite domains*, Dokl. Akad. Nauk SSSR, 286 (1986), pp. 308–311. (In Russian.)

[6] O. Belegradek, A. Stolboushkin, and M. Taitslin, *On Order-Generic Queries*, Tech. Report 96-01, DIMACS, Rutgers University, New Brunswick, NJ, 1996.

[7] M. Benedikt, G. Dong, L. Libkin, and L. Wong, *Relational expressive power of constraint query languages*, in Proc. 15th ACM Symposium on Principles of Database Systems, ACM Press, New York, 1996, pp. 5–16.

[8] M. Benedikt and L. Libkin, *On the structure of queries in constraint query languages*, in Proc. 11th IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 25–34.

[9] F. Benhamon and A. Colmerauer, eds., *Constraint Logic Programming: Selected Research*, MIT Press, Cambridge, MA, 1993.

[10] J. Bochnak, M. Coste, and M.-F. Roy, *Géométrie algébrique réelle*, Springer-Verlag, Berlin, 1987.

[11] A. Chandra and D. Harel, *Computable queries for relational data bases*, J. Comput. System Sci., 21 (1980), pp. 156–178.

[12] G. Collins, *Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition*, Lecture Notes in Computer Science 33, Springer-Verlag, New York, 1975, pp. 134–183.

[13] H. Enderton, *A Mathematical Introduction to Logic*, Academic Press, New York, 1972.

[14] S. Grumbach, J. Su, and C. Tollu, *Linear constraint query languages: Expressive power and complexity*, in Logic and Computational Complexity, D. Leivant, ed., Lecture Notes in Computer Science 960, Springer-Verlag, New York, 1995, pp. 426–446.

[15] R. Hull and J. Su, *Domain independence and the relational calculus*, Acta Inform., 31 (1994), pp. 513–524.

[16] P. Kanellakis, G. Kuper, and P. Revesz, *Constraint query languages*, J. Comput. System Sci., 51 (1995), pp. 26–52.

[17] G. Kuper, *On the expressive power of the relational calculus with arithmetic constraints*, in ICDT'90, Lecture Notes in Computer Science 470, S. Abiteboul and P. Kanellakis, eds., Springer-Verlag, New York, 1990, pp. 202–214.

[18] M. Otto and J. Van den Bussche, *First-order queries on databases embedded in an infinite structure*, Inform. Process. Lett., 60 (1996), pp. 37–41.

[19] J. Paredaens, J. Van den Bussche, and D. Van Gucht, *Towards a theory of spatial database queries*, in Proc. 13th ACM Symposium on Principles of Database Systems, ACM Press, New York, 1994, pp. 279–288.

[20] J. Paredaens, J. Van den Bussche, and D. Van Gucht, *First-order queries on finite structures over the reals (extended abstract)*, in Proc. 10th IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 79–87.

[21] A. Stolboushkin and M. Taitslin, *Linear vs. order constraints over rational databases*, in Proc. 15th ACM Symposium on Principles of Database Systems, ACM Press, New York, 1996, pp. 17–27.

[22] L. Van Den Dries, *Alfred Tarski's elimination theory for real closed fields*, J. Symbol. Logic, 53 (1988), pp. 7–19.

SIAM J. COMPUT.
Vol. 27, No. 6, pp. 1764–1811, December 1998
© 1998 Society for Industrial and Applied Mathematics
012

SPIRALITY AND OPTIMAL ORTHOGONAL DRAWINGS*

GIUSEPPE DI BATTISTA†, GIUSEPPE LIOTTA‡, AND FRANCESCO VARGIU§

Abstract. We deal with the problem of constructing the orthogonal drawing of a graph with the minimum number of bends along the edges. The problem has been recently shown to be NP-complete in the general case. In this paper we introduce and study the new concept of spirality, which is a measure of how an orthogonal drawing is "rolled up," and develop a theory on the interplay between spirality and number of bends of orthogonal drawings. We exploit this theory to present polynomial time algorithms for two significant classes of graphs: series-parallel graphs and 3-planar graphs. Series-parallel graphs arise in a variety of problems such as scheduling, electrical networks, data-flow analysis, database logic programs, and circuit layout. Also, they play a central role in planarity problems. Furthermore, drawings of 3-planar graphs are a classical field of investigation.

Key words. graph drawing, orthogonal representation, planar embedding, bend minimization

AMS subject classifications. 05C85, 90B10, 90C27

PII. S0097539794262847

1. Introduction.

A graph drawing algorithm receives as input a graph and produces as output a drawing that nicely represents such a graph; several references on the subject of graph drawing can be found in [23, 7]. Most graph drawing algorithms can be roughly split into the following two main steps.

1. A planar embedding of the given graph is found by a *planarization algorithm*, possibly by inserting dummy vertices for crossings. The planar embedding is usually described by the cyclic ordering of the edges incident at each vertex. Planarization algorithms are implemented by using variations of the classical planarity testing algorithms (see, e.g., [12]).
2. Once a planar embedding has been found, a *representation algorithm* is applied to produce the final drawing. Such an algorithm is selected depending on the requirements of the application and on the graphic standard. It can be targeted to minimize the global area of the drawing, to have as few bends as possible along the edges, to emphasize symmetries, etc.

The representation algorithm produces a drawing within the planar embedding computed by the planarization algorithm. However, the choice of the planar embedding can deeply affect the results obtained by the representation algorithm. In Fig. 1 we show two different planar embeddings of the same graph. Besides each planar embedding we show the *orthogonal drawing* (edges are mapped to polygonal chains of horizontal and vertical segments) with the minimum number of bends that can be

* Received by the editors February 4, 1994; accepted for publication (in revised form) October 15, 1996; published electronically June 3, 1998. This research was partially supported by CNR under grant CTB 94.00023.07, by NATO-CNR Advanced Fellowship Programme, the National Science Foundation under grant CCR-9423847, the EC ESPRIT Long Term Research Project ALCOM-IT under contract 20244, and the U.S. Army Research Office under grant DAAH04–96–1–0013.
http://www.siam.org/journals/sicomp/27-6/26284.html

† Dipartimento di Informatica e Automazione, Università di Roma Tre, via della Vasca Navale 84, I-00146 Roma, Italia (dibattista@iasi.rm.cnr.it).

‡ Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza," via Salaria 113, I-00198 Roma, Italia (liotta@dis.uniroma1.it). Part of this research was done while the author was with the Center of Geometric Computing at the Department of Computer Science, Brown University.

§ Autorità per l'Informatica nella Pubblica Amministrazione, piazzale Kennedy 20, I-00144 Roma, Italia (vargiu@aipa.it).

constructed preserving that embedding. In the drawing of Fig. 1a we have $n/3 - 2$ bends (where n is the number of vertices); in the one of Fig. 1b we have no bends.

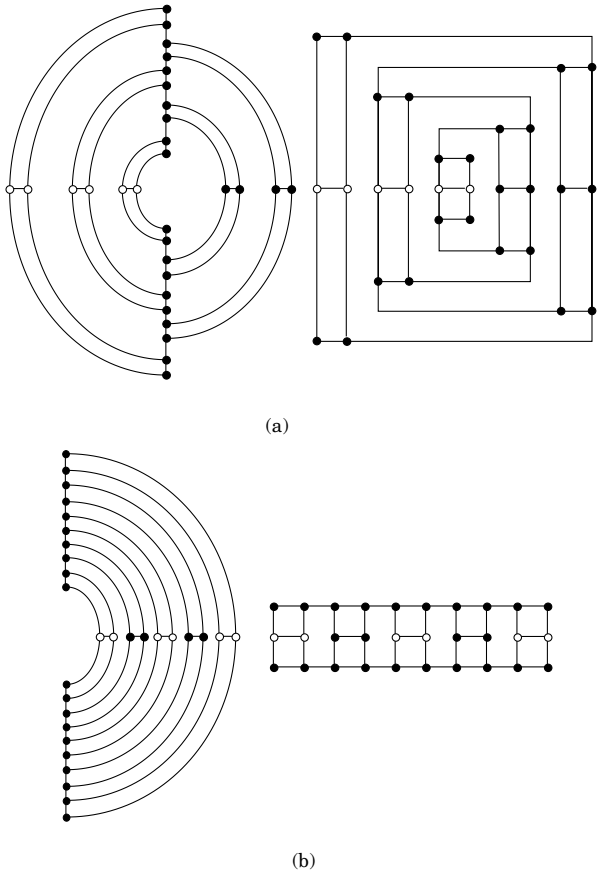


FIG. 1. *The choice of the embedding deeply affects the number of bends in the orthogonal drawing.*

Thus, it naturally raises the problem of choosing, in the planarization algorithm, the "best" embedding from the representation algorithm point of view. Although the problem is quite natural there are only a few contributions on this topic; observe that a planar graph has (in general) an exponential number of embeddings. To give an example, the problem of constructing straight-line upward drawings of series-parallel digraphs without a fixed embedding has been addressed in [2] (an upward drawing is such that all the edges follow monotonically the vertical axis); in [2] it is shown that fixed-embedding drawing strategies can lead to straight-line upward drawings of series-parallel digraphs with exponential area, and variable-embedding algorithms are needed to achieve optimal area.

In this paper we deal with the classical problem of constructing orthogonal drawings with the minimum number of bends along the edges. Valiant [29] showed that a graph has an orthogonal drawing if and only if it is 4-planar (a graph is k-planar if it is planar and each vertex has degree at most k). Tamassia [21] proposed a very elegant representation algorithm that solves the problem in polynomial time for graphs with a fixed embedding. The algorithm is based on a combinatorial characterization that

allows to map the problem into a min-cost flow one. The result of Tamassia disproves a conjecture of Storer [19] that the problem is NP-hard. Linear time heuristics for the same problem were proposed by Tamassia and Tollis [24, 25], Liu, Morgana, and Simeone [16], Kant [13], Biedl and Kant [3], and Papakostas and Tollis [18]. Tamassia, Tollis, and Vitter [26, 27] gave lower bounds for the problem and the first parallel algorithm. A brief survey of orthogonal drawings can be found in [22].

However, all the above papers deal with fixed-embedding graphs. The problem of finding the planar embedding that leads to the minimum number of bends has been recently shown to be NP-complete [11]. In this paper we show that the problem can be solved by polynomial time algorithms for two significant classes of graphs. A list of the main results of the paper follows.

- We introduce and study the new concept of *spirality*, that is, a measure of how an orthogonal drawing is "rolled up."
- We develop a theory on the interplay between spirality and the number of bends in orthogonal drawings.
- We apply the above theory to show that the problem of finding a planar embedding that leads to an orthogonal drawing with the minimum number of bends can be solved in polynomial time for 3-planar graphs and for series-parallel graphs.

Also, we show how the time bound for series-parallel graphs can be reduced in the case when the graph is 3-planar.

Series-parallel graphs arise in a variety of problems such as scheduling, electrical networks, data-flow analysis, database logic programs, and circuit layout. Also, they play a central role in planarity problems [20, 28, 5, 6]. In Fig. 2 we show two different planar embeddings of the same series-parallel graph. Besides each planar embedding we show the orthogonal drawing with the minimum number of bends that can be constructed preserving that embedding. In the drawing of Fig. 2a we have almost twice the number of bends as in that of Fig. 2b. Also, drawings of 3-planar graphs are a classical field of investigation (see, e.g., [13]). Observe that the graph of Fig. 1 is 3-planar.

Our algorithms exploit the properties of spirality, min-cost flow techniques, and a variation of the $SPQR$ trees [5, 6], a data structure that implicitly represents all the planar embeddings of a planar graph. Observe that a different concept of spirality has already been introduced in the literature for studying the properties of polygons. (See, e.g., [9].)

The paper is organized as follows. Preliminaries are in section 2. The general approach of the paper is briefly described in section 3. The concept of spirality of an orthogonal drawing and the relationships between spirality and number of bends are studied in section 4. In sections 5 and 6 we present polynomial time algorithms and a new data structure for computing optimal orthogonal drawings of 3-planar graphs. The algorithms presented in those sections are detailed enough to be implemented with limited effort. The results of sections 5 and 6 are extended to series-parallel graphs in section 7. Finally, open problems are listed in section 8. Some of the proofs that are conceptually straightforward but involve tedious case analyses are omitted and can be found in [4].

2. Preliminaries. First, we briefly review some definitions of connectivity and planarity [8, 17]; second, we give some properties of orthogonal drawings and orthogonal representations; third, we define SPQ^*R trees; and finally, we reiterate basic definitions of flow networks.

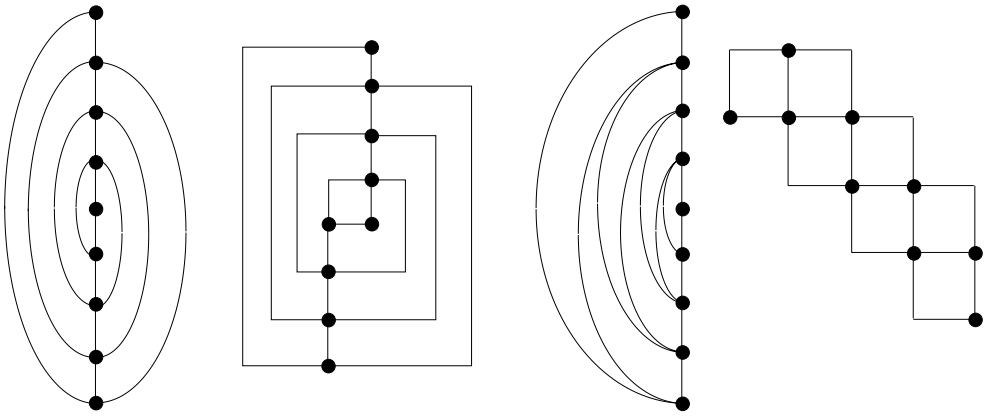


FIG. 2. *Two embeddings of a series-parallel graph that give rise to two optimal orthogonal drawings with different numbers of bends.*

Several terms are defined throughout the remainder of the paper, after these preliminaries. For the reader's convenience we list all such terms in the appendix.

2.1. Connectivity and planarity. A separating k-set of a graph G is a set of k vertices whose removal increases the number of connected components of G. Separating 1-sets (2-sets) are called *cutvertices* (*separation pairs*). A connected graph with no cutvertices is a 2-*connected graph*.

Let Γ be a *planar drawing* of a graph G. Γ maps each vertex of G to a distinct point of the plane, and each edge (u, v) to a simple Jordan curve with endpoints u and v; no two edges intersect, except at common endpoints. A planar drawing Γ divides the plane into topologically connected regions called *faces* of Γ; each face is identified by the circular list of the vertices and the edges of its boundary. The unbounded region is referred to as the *external face* of Γ. A graph is *planar* if it has a planar drawing. A planar graph is k-*planar* if its vertices have degree at most k.

A *series-parallel graph* is recursively defined as follows. A simple cycle with three edges is a series-parallel graph. The graph obtained by splitting an edge of a series-parallel graph into two edges is a series-parallel graph. The graph obtained by inserting an edge between a separation pair $\{u, w\}$ of a series-parallel graph is a series-parallel graph. It is trivial to see that series-parallel graphs defined in this way are 2-connected and planar.

Two planar drawings Γ' and Γ'' of a planar graph are *equivalent* when, for each vertex v, (1) Γ' and Γ'' have the same circular clockwise ordering of the edges incident on v and of the faces around v; (2) Γ' and Γ'' have the same external face. In this way the planar drawings of a planar graph are grouped into equivalence classes. An *embedding* of a planar graph G corresponds to an equivalence class of planar drawings of G.

A planar graph G with a given embedding Ψ is an *embedded graph*. A circular list of edges and vertices of G that corresponds to a face of any drawing of Ψ is a *face* of G. The face of G that corresponds to the external face of any drawing of Ψ is the *external face* of G. The *adjacency list* of each vertex v of an embedded graph is a clockwise ordered sequence of the alternate faces and edges around v.

2.2. Decomposition trees. A *split pair* of G is either a separation pair of G or a pair of adjacent vertices. A *split component* of a split pair $\{u, w\}$ is either an edge (u, w) or a maximal subgraph G_{uw} of G such that $\{u, w\}$ is not a split pair of G_{uw}. Vertices u and w are the *poles* of the split component. In the following we often denote by G_{uw} a split component whose poles are u and w.

Let G be an embedded graph and $G_{uw} \subset G$ be a split component of G. The edges of G that are incident on a pole v ($v = u, w$) of G_{uw} and that belong (do not belong) to G_{uw} are called *internal edges* (*external edges*) of v with respect to G_{uw}; the number of such edges is called *internal degree* (*external degree*) of v with respect to G_{uw}. Let f' and f'' be the two faces such that (i) f' and f'' do not belong to G_{uw}; (ii) f' (f'') shares with G_{uw} a path P'_{uw} (P''_{uw}) from u to w; (iii) when going around f' in the positive direction, P'_{uw} is traversed from u to w. Face f' is called the *right face* of G_{uw} and face f'' is called the *left face* of G_{uw}. Nodes u and w divide a face f_e of G_{uw} into two simple paths. The *right path* is the one traversed from u to w when going around f_e in the positive direction; the *left path* is the other one.

Let G be a planar 2-connected graph and let (s, t) be an edge of G called the *reference edge*. An *SPQ^*R tree* T of G (a simple variation of the $SPQR$ trees introduced in [5]) is a tree describing a recursive decomposition of G with respect to its split pairs, and it is used to synthetically represent all the embeddings of G with (s, t) on the external face (we always assume, unless stated otherwise, that (s, t) is on the external face). Nodes of T are of four types: S, P, Q^*, and R. Each node μ has an associated graph called the *skeleton* of μ and denoted by $skeleton(\mu)$. Nonroot nodes of T are called *internal nodes*. Starting from the reference edge, T is recursively defined as follows.

Chain case: If G consists of a simple path from s to t then T is a single Q^*-node μ whose skeleton is G itself.

Series case: If G is 1-connected, let $c_1, \ldots, c_{k-1}$ ($k \geq 2$) be the cutvertices of G such that no cutvertex has degree less than 3; let $c_1, \ldots, c_{k-1}$ partition G into graphs $G_1, \ldots, G_k$. The root of T is an S-node μ. Graph $skeleton(\mu)$ is the chain $e_1, \ldots, e_k$, where edge e_i goes from c_{i-1} to c_i, $c_0 = s$, and $c_k = t$.

Parallel case: If s and t are a split pair for G with split components $G_1, \ldots, G_k$ ($k \geq 2$), the root of T is a P-node μ. Graph $skeleton(\mu)$ consists of k parallel edges from s to t, denoted $e_1, \ldots, e_k$.

Rigid case: If none of the above cases applies, let $\{s_1, t_1\}, \ldots, \{s_k, t_k\}$ be the maximal split pairs of G ($k \leq 1$), and for $i = 1, \ldots, k$, let G_i be the union of all the split components of $\{s_i, t_i\}$. The root of T is an R-node μ. Graph $skeleton(\mu)$ is obtained from G by replacing each subgraph G_i with edge e_i from s_i to t_i.

We call the *pertinent graph* of μ the graph whose decomposition tree is the subtree rooted at μ. Also, the *virtual edge* of μ is the edge representing the pertinent graph of μ in the skeleton of its parent.

Let G be a planar 2-connected graph; we define an SPQ^*R tree T of G such that:

- the root of T is a P-node with two children, one of which is the reference edge (s, t);
- each internal P-node of T has children R-, S-, or Q^*-nodes;
- each S-node of T has two children.

We call T the *canonical decomposition tree* of G. An example of a canonical decomposition tree is given in Fig. 3, where the skeleton of an internal R-node is in evidence; the reference edge is $(1, 10)$. ξ-labels associated with some of the nodes will

be used later in the paper. It is easily proved that such a decomposition tree always exists for a 2-connected graph.

Since in the following we refer only to canonical decomposition trees, we briefly call them *decomposition trees*. Also, H_μ denotes an orthogonal representation of the pertinent graph of μ.

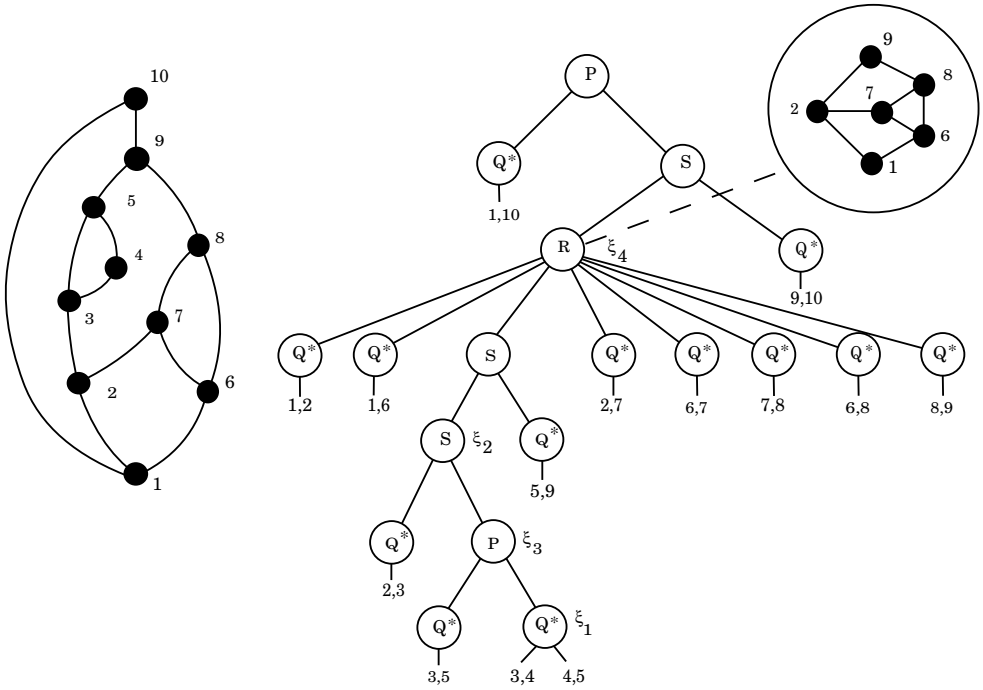


FIG. 3. *A canonical decomposition tree.*

PROPERTY 2.1. *The decomposition tree of a series-parallel graph can only contain S-nodes, P-nodes, and Q*-nodes.*

The following property can be easily proved by using arguments based on the degree of the vertices.

PROPERTY 2.2. *Let T be the decomposition tree of a 3-planar graph and let μ be an internal node of T.*

1. *If μ is an S-node and one of its children is either an R-node or a P-node, then the other child of μ can be either a Q^*-node or an S-node.*
2. *If μ is a P-node, then it has two children.*
3. *If μ is a P-node or an R-node, then its children are Q^*-nodes or S-nodes.*

2.3. Orthogonal drawings and orthogonal representations. A planar drawing of a planar graph G such that all edges of G are mapped to polygonal chains of horizontal and vertical segments is an *orthogonal drawing* of G. Examples of orthogonal drawings are in Fig. 4. A planar graph has a orthogonal drawing if and only if it is 4-planar [29].

Two equivalent orthogonal drawings of an embedded graph are *shape-equivalent* when (1) for each vertex v, consecutive edges in the adjacency list of v form the same angle in the two drawings; and (2) for each edge (u, v), following from u to v the

polygonal chain representing (u, v), we have the same (possibly empty) sequence of left and right turns in the two drawings. For example, the orthogonal drawings of Fig. 4 are shape-equivalent.

A *labeled embedded graph* H is an embedded graph defined as follows. (1) Each edge e is associated to a (possibly empty) sequence of L- and R-symbols. When walking along $e = (u, v)$, such a sequence is read in two different ways depending on the chosen direction. Suppose, when walking from u to v, that such a sequence is read σ; then when walking from v to u it is read σ', where σ' is obtained by reversing σ and by exchanging L- and R-symbols. (2) Each face f in the adjacency list of vertex v has (possibly) a label in $\{R,\ L, LL\}$.

An *orthogonal graph* H is a labeled embedded graph that describes a class of shape-equivalent orthogonal drawings; it is defined as follows.

- Each edge (u, v) of H is associated with a (possibly empty) sequence of L- and R-symbols, each specifying a left or right turn that is found following (u, v) from u to v.
- For each vertex v of H, the label associated with each face f in the adjacency list of v specifies what happens when we go through v, walking around f in the positive direction (i.e., having f at one's right) in any drawing of the shape-equivalent class. Namely, (1) f is labeled L if we turn left; (2) f is labeled R if we turn right; (3) f is labeled LL when v has degree 1 and we have to come back ("two turns left are enough to come back"); (4) f is not labeled when we go straight.

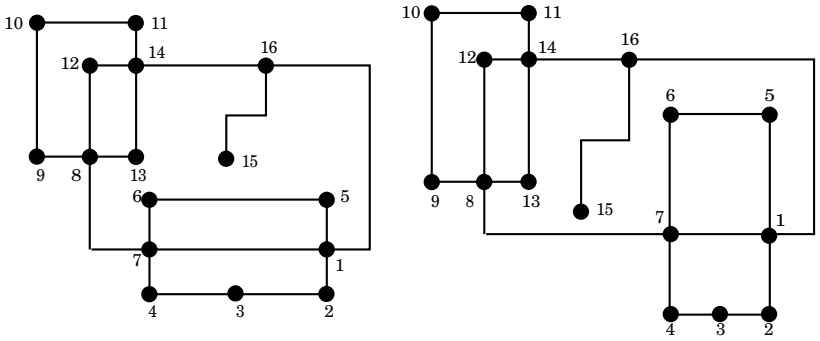


FIG. 4. *Two shape-equivalent orthogonal drawings of the same graph.*

In Fig. 5 we show the orthogonal graph that describes the orthogonal drawings of Fig. 4; besides each edge we report both the sequences that are read by walking on the edge in the two possible directions.

We denote by $\deg(v)$ the degree of vertex v. The following property characterizes the orthogonal graphs; it has been proved in [21].

PROPERTY 2.3. *A labeled embedded graph H is an orthogonal graph if and only if the following two conditions hold.*

1. *For each vertex v of H, let $|R_v|$ ($|L_v|$) be the number of R-symbols (L-symbols) of the labels associated with the faces that appear in the adjacency list of v. Then,*

$$|R_v| - |L_v| = 2\deg(v) - 4.$$

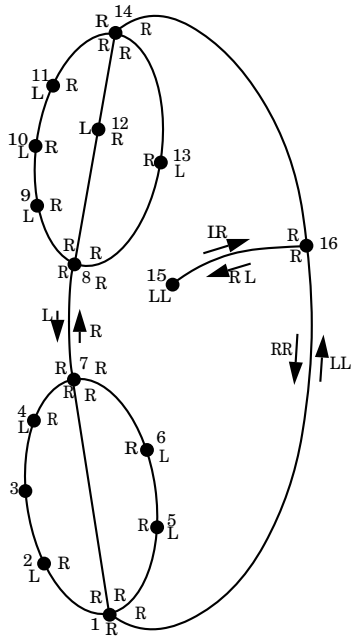


FIG. 5. *The orthogonal graph describing the orthogonal drawings of Fig.* 4.

2. *For each face f of H suppose we walk around f in the positive direction. Let $|R_e|$ ($|L_e|$) be the total number of R-symbols (L-symbols) that are encountered by traversing the edges of f. Let $|R_f|$ ($|L_f|$) be the number of R-symbols (L-symbols) of the labels associated with f in the adjacency lists of vertices of f. Then*

$$|R_e| + |R_f| - |L_e| - |L_f| = \pm 4,$$

where the plus sign holds if f is an internal face, and the minus sign, if f is the external face.

Proof. For the sake of completeness we briefly sketch the proof (see [21]). Part 1 is proved by observing that $|R_v|$ ($|L_v|$) is the number of $\pi/2$ ($3\pi/2$) angles around v and that such angles sum up to 2π. Part 2 is proved by observing that $|R_e|$ ($|L_e|$) is the number of $\pi/2$ ($-\pi/2$) angles along the edges of f, that $|R_f|$ ($|L_f|$) is the number of $\pi/2$ ($-\pi/2$) angles on the vertices of f, and that such angles sum up to $\pm 2\pi$, depending on whether we consider the interior or the exterior part of the face. □

The above property can be easily verified on the graph of Fig. 5. For example, it is easy to see that Condition 1 holds for vertex 1; namely, it has degree 4 and the labels of the faces that appear in its adjacency list are R-symbols.

From Property 2.3 it directly descends the following property.

PROPERTY 2.4. *Let C be a simple cycle of an orthogonal graph. The number of right turns minus the number of left turns encountered by walking clockwise around C is four.*

For example, the cycle composed by vertices 1, 2, 3, 4, 7, 6, 5 of Fig. 5 verifies Property 2.4.

An *orthogonal representation* of a graph G is an orthogonal graph with the same vertices and edges of G. The *cost* of an orthogonal representation is the number of its

bends. An *optimal orthogonal representation* of G is an orthogonal representation of G with the minimum cost. An *optimal orthogonal drawing* of G is an orthogonal drawing of the shape-equivalence class described by an optimal orthogonal representation of G.

Orthogonal graphs have also been studied by Vijayan and Wigderson in [30].

2.4. The min-cost flow problem. We briefly recall some definitions of flow networks. A *network* N consists of (1) a finite set of nodes including exactly one *source* node s and one *sink* node t; (2) a set of oriented edges (in what follows, *arcs*), labeled with a nonnegative *capacity*, a nonnegative *lower bound*, and a nonnegative *cost function*.

A *flow* x in N associates to each arc a a nonnegative number $x(a)$; flow $x(a)$ cannot exceed the capacity of a and cannot be less than the lower bound of a. Also, for each node $v \neq s, t$ of N, the sum of the flows of the incoming arcs is equal to the sum of the flows of the outgoing arcs. Let c_a be the cost function of arc a; the *cost* of the flow in a is $c_a(x(a))$. The *cost of the flow* x in N is the sum of the costs of the flow in the arcs of N. The *value of the flow* in N is the sum of the flows of the outgoing arcs of s. A *min-cost flow problem* is stated as follows: given a network N and a positive number z, find a flow x in N such that the value of x is z and the cost of the flow is minimum. The min-cost flow problem has been intensively studied in the literature (see, e.g., [10, 15, 1]).

3. The general approach. The general approach underlying the paper can be summarized as follows. Let us concentrate on the case of 3-planar 2-connected graphs. We shall see that the results on such graphs can be easily extended to general 3-planar graphs and to 4-planar series-parallel graphs.

To compute an optimal orthogonal representation of a 3-planar 2-connected graph G, the first step is to construct a decomposition tree T of G. The optimal orthogonal representation of G is then built by visiting T from bottom to top. In fact, we show that the optimal orthogonal representation of the pertinent graph of a node μ of T can be constructed by considering and composing a limited set of orthogonal representations (not necessarily optimal) of the pertinent graphs of the children $\mu_1, \ldots, \mu_k$ of μ; we call such limited sets of orthogonal representations *optimal sets*.

If μ is an S-node or a P-node, then an optimal orthogonal representation of the pertinent graph of μ can be built in polynomial time in two steps. In the first step one orthogonal representation for each of the optimal sets of $\mu_1, \ldots, \mu_k$ is suitably selected. In the second step the selected representations are assembled into a representation of the pertinent graph of μ.

Otherwise (μ is an R-node), the optimal orthogonal representation of μ can be computed with a network-flow technique; this is done in polynomial time by exploiting certain convexity properties that hold for the optimal sets of $\mu_1, \ldots, \mu_k$.

Thus, for 2-connected 3-planar graphs we have algorithms that, from bottom to top, compute in polynomial time the optimal sets of all the nodes of T.

The conceptual tool that is used to select, among the infinite possible orthogonal representations of the pertinent graphs of the nodes of T, those that are needed in the optimal sets is the spirality. As we said in the introduction, the spirality measures how much an orthogonal representation is "rolled up." In particular, we exploit two basic properties of the spirality that can be roughly stated as follows.

- The split components of an optimal orthogonal representation cannot be rolled up too much. Intuitively, if too many turns are done, then too many bends are spent.

- Two orthogonal representations of the same split component and with the same spirality can replace each other into an orthogonal representation of G.

4. Spines and spirality. In this section we define and study an invariant, called *spirality*, of the equivalence class of drawings described by a component of an orthogonal graph. Unless stated otherwise we suppose that all the graphs mentioned from now on are 2-connected.

4.1. Spines. Let H be an orthogonal graph and $H_{uw} \subset H$ be a split component of H. A pole v ($v = u$ or $v = w$) of H_{uw} is called a *bridge pole* when its internal degree with respect to H_{uw} is 1; v is called a *nonbridge pole* when its internal degree with respect to H_{uw} is greater than 1.

In order to define the spirality of H_{uw} we add to H some dummy vertices called *alias vertices* of the poles u, w. Three cases are possible.

1. Let v be a bridge pole of H_{uw}; the *alias vertex* v' of v coincides with v.
2. Let v be a nonbridge pole of H_{uw} with external degree 1; the *alias vertex* v' of v is defined as follows. Let $e_{out} = (x, v)$ be the external edge of v with respect to H_{uw}; e_{out} is substituted by a new vertex v' and two new edges (x, v'), (v', v). The labeling of (x, v') is the same as that of e_{out}, (v', v) has no labels, and the two faces around v' have no labels in the adjacency list of v'. Intuitively, we have cut the first segment of the drawing of e_{out} into two pieces.
3. Let v be a nonbridge pole of H_{uw} with external degree 2, and let e'_{out} and e''_{out} be the external edges of v; the *alias vertices* of v are a pair v', v'' of dummy vertices obtained by cutting e'_{out} and e''_{out}, respectively. They are defined as above.

In Fig. 6 we show an orthogonal drawing of an orthogonal graph with, in evidence, a split component H_{uw}; vertices w' and w'' are the alias vertices of w, and vertex u' is the alias vertex of u.

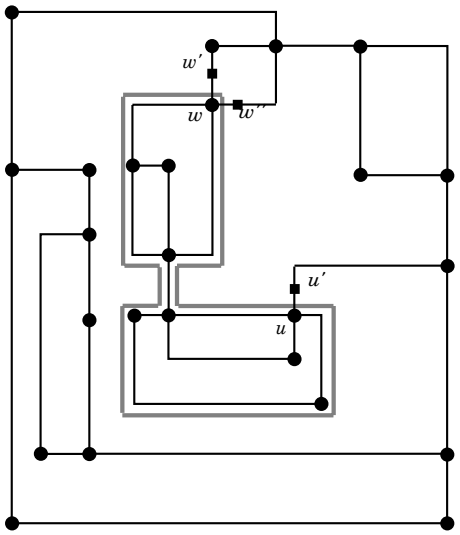


FIG. 6. *Adding alias vertices.*

DEFINITION 4.1. *Let H be an orthogonal graph and let $H_{uw} \subset H$ be a split component of H. Let u' be an alias vertex of u and let w' be an alias vertex of w. Let P_{uw} be a simple path in H_{uw} from u to w. A* spine $S_{u'w'}$ *of H_{uw} is the simple path obtained by concatenating edge (u', u), path P_{uw}, and edge (w, w').*

In Figs. 7a and 7b we show two spines of the split component from the example of Fig. 6.

Let v_1 and v_2 be two vertices of H, and $P_{v_1v_2}$ be a simple path between v_1 and v_2. We associate with $P_{v_1v_2}$ an integer $n(P_{v_1v_2})$ defined as the number of right turns minus the number of left turns found when going along $P_{v_1v_2}$ from v_1 to v_2. In Fig. 7a we have $n(S_{u'w'}) = 2$; in Fig. 7b, $n(S_{u'w''}) = 3$.

LEMMA 4.1. *Let H be an orthogonal graph and let $H_{uw} \subset H$ be a split component of H. Let u' be an alias vertex of u and let w' be an alias vertex of w. Let $S'_{u'w'}$ and $S''_{u'w'}$ be two spines of H_{uw}; we have $n(S'_{u'w'}) = n(S''_{u'w'})$.*

Proof. Let $P_{u'w'}$ be a simple path in H that joins u' and w', such that no edge of $P_{u'w'}$ belongs to H_{uw}. Let C' and C'' be two simple cycles such that C' is composed by $S'_{u'w'}$ and $P'_{u'w'}$ and C'' is composed by $S''_{u'w'}$ and $P'_{u'w'}$. By Property 2.4 applied to C' we have $n(S'_{u'w'}) - n(P'_{u'w'}) + a = 4$, where a is the number of right turns minus the number of left turns encountered at u' and w', when going clockwise around C'; observe that a can be different from 0 only when the poles are bridge poles. By applying the same argument to C'' it follows that $n(S''_{u'w'}) = n(S'_{u'w'})$. □

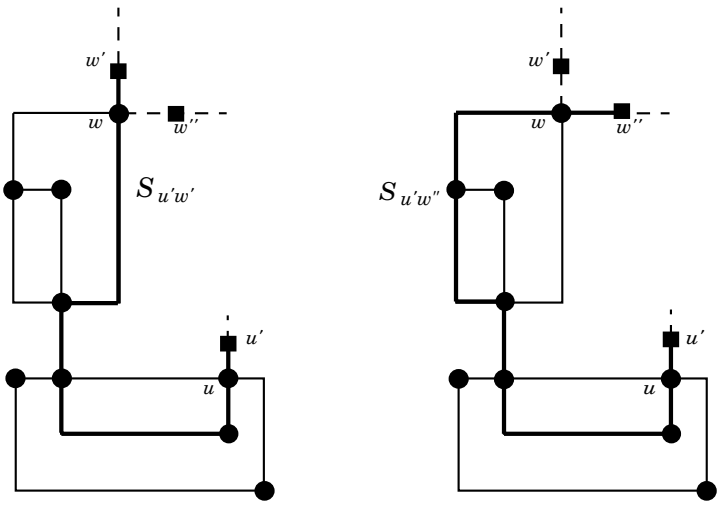


FIG. 7. *Two spines of the split component highlighted in Fig.* 6.

4.2. Spirality.

DEFINITION 4.2. *Let H be an orthogonal graph and $H_{uw} \subset H$ be a split component of H. The* spirality $\sigma_{H_{uw}}$ *of H_{uw} is defined as follows. Three cases are possible, depending on the number of alias vertices of u and w.*

1. *Both u and w have just one alias vertex, u' and w', respectively. Let $S_{u'w'}$ be a spine of H_{uw}; $\sigma_{H_{uw}} = n(S_{u'w'})$.*

2. *Pole u has just one alias vertex u'; w has two alias vertices w' and w''. Let $S_{u'w'}$ and $S_{u'w''}$ be two spines of H_{uw}; $\sigma_{H_{uw}} = (n(S_{u'w'}) + n(S_{u'w''}))/2$.*
3. *Pole u has two alias vertices u' and u''; w has two alias vertices w' and w''. Suppose u' and w' are on the same face of H. Let $S_{u'w'}$ and $S_{u''w''}$ be two spines of H_{uw}; $\sigma_{H_{uw}} = (n(S_{u'w'}) + n(S_{u''w''}))/2$.*

For example, the spirality of the split component in evidence in Fig. 6 is $\sigma_{H_{uw}} = 5/2$ (see also Figs. 7a and 7b).

PROPERTY 4.1. *If the poles of H_{uw} satisfy case 2 of Definition 4.2, then $2\sigma_{H_{uw}}$ is an odd integer number; if the poles of H_{uw} satisfy either case 1 or case 3 of Definition 4.2, then $\sigma_{H_{uw}}$ is an integer number.*

Proof. We prove the first case; the proof of the second case is analogous. Let u' be the alias vertex of u and let w', w'' be the alias vertices of w. Let $P_{u,w}$ be a simple path of H_{uw} from u to w. Let $S_{u'w'}$ be a simple path composed by edge (u', u), path $P_{u,w}$, and edge (w, w'); let $S_{u'w''}$ be a simple path composed by edge (u', u), path $P_{u,w}$, and edge (w, w''); $S_{u'w'}$ and $S_{u'w''}$ are two spines of H_{uw} that differ only for one turn at vertex w. Thus, $n(S_{u'w'}) + n(S_{u'w''})$ is an odd integer. □

The following property shows that the spirality of a split component H_{uw} depends only on the shape of H_{uw} and, possibly, on the labels associated with the right and left faces in the adjacency lists of the poles.

PROPERTY 4.2. *Let H and H' be two orthogonal graphs that are orthogonal representations of the same graph G; let G_{uw} be a split component of G. Suppose that (1) the orthogonal representation H_{uw} of G_{uw} is the same in H and in H'; (2) the label associated with the right (left) face of H_{uw} in the adjacency list of each pole of H_{uw} that is not a bridge pole is the same in H and in H'. The spirality $\sigma_{H_{uw}}$ is the same in H and in H'.*

Proof. From Definition 4.1 any spine S of H_{uw} is composed by a simple path P_{uw} of H_{uw} from u to w, and two edges (v, v') where v is a pole of H_{uw} and v' is an alias vertex of that pole ($v \equiv v'$ if v is a bridge pole). Since each edge (v, v') has no bends, to evaluate $n(S)$ we need $n(P_{uw})$ and information about possible turns encountered at poles when going along S; but this information is univocally given by the labels associated with the right and left faces of H_{uw} in the adjacency lists of the poles. □

4.3. Spirality of a series. Let H be an orthogonal graph and T be the decomposition tree of H.

Let μ be an internal S-node of T, with children μ_1 and μ_2; let $H_\mu \subset H$ be the pertinent orthogonal graph of μ and let $H_{\mu_i} \subset H_\mu$ be the pertinent orthogonal graph of μ_i $(i = 1, 2)$. In Fig. 8 we show the series of the two components in the subgraph in evidence in Fig. 6.

LEMMA 4.2. *The spirality σ_{H_μ} of H_μ is related to the spiralities $\sigma_{H_{\mu_i}}$ of the H_{μ_i} $(i = 1, 2)$ by*

$$\sigma_{H_\mu} = \sigma_{H_{\mu_1}} + \sigma_{H_{\mu_2}}.$$

Proof. The proof is illustrated in Fig. 9. Let u, w be the poles of H_μ, let u_1, w_1 be the poles of H_{μ_1}, and let u_2, w_2 be the poles of H_{μ_2}. Assume $u_1 \equiv u$, $w_1 \equiv u_2$, and $w_2 \equiv w$.

Several cases are possible, according to the different types of poles of H_{μ_1} and H_{μ_2} and according to the external degree of the poles of H_μ.

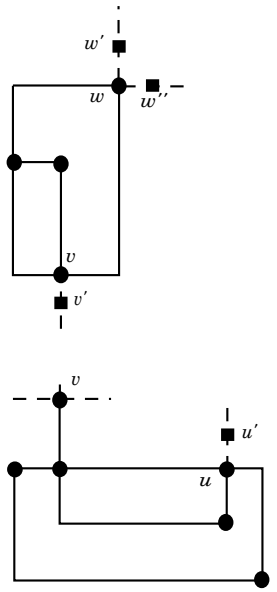


FIG. 8. *The series of two split components of the subgraph highlighted in Fig.* 6.

1. Pole w_1 is a nonbridge pole in H_{μ_1} and pole u_2 is a bridge pole in H_{μ_2}. Since u_2 is a bridge pole in H_{μ_2}, the alias vertex of u_2 is the pole itself. Also, let w'_1 be the alias vertex of w_1 on the edge of H_{μ_2} incident on w_1. We distinguish two subcases, according to the external degree of u and w in H_μ.
   (a) $u_1 \equiv u$ and $w_2 \equiv w$ are bridge poles or they have external degree 1 in H_μ (see Fig. 9a). In this case just one spine is needed to evaluate the spirality of H_μ, H_{μ_1}, and H_{μ_2}. Let u'_1 be the alias vertex of u_1 with respect to H_{μ_1}, and let w'_2 be the alias vertex of w_2 with respect to H_{μ_2}; u'_1 and w'_2 coincide with the alias vertices of u and w with respect to H_μ. Let $S_{u'_1 w'_1}$ be a spine of H_{μ_1}, and let $S_{u_2 w'_2}$ be a spine of H_{μ_2}. Let $P_{w'_1 w'_2}$ be the subpath of $S_{u_2 w'_2}$ from w'_1 to w'_2; note that $n(P_{w'_1 w'_2}) = n(S_{u_2 w'_2})$ from the definition of alias vertices. Let $S_{u'_1 w'_2}$ be a path in H_μ composed by spine $S_{u'_1 w'_1}$ and path $P_{w'_1 w'_2}$. $S_{u'_1 w'_2}$ is a spine of H_μ and $n(S_{u'_1 w'_2}) = n(S_{u'_1 w'_1}) + n(P_{w'_1 w'_2})$. It follows that $\sigma_{H_\mu} = \sigma_{H_{\mu_1}} + \sigma_{H_{\mu_2}}$.
   (b) $w_2 \equiv w$ is a nonbridge pole with external degree 2 in H_μ (see Figs. 9b and 9c). In this case two spines are requested to evaluate the spirality of H_μ. With analogous reasoning as in case (a), we can define the spines of H_μ starting from the spines of H_{μ_1} and H_{μ_2}; the rest of the proof is a simple variation of the previous case.
2. w_1 is a nonbridge pole in H_{μ_1} and u_2 is a nonbridge pole in H_{μ_2} (the total degree of $w_1 \equiv u_2$ is four). Let w'_1, w''_1 be the alias vertices of w_1 with respect to H_{μ_1}, and let u'_2, u''_2 be the alias vertices of u_2 with respect to H_{μ_2} such that w'_1 and u'_2 are in the same face of H, say, the right face of H_μ. We distinguish two subcases, according to the external degree of u and w in H_μ.
   (a) Both $u_1 \equiv u$ and $w_2 \equiv w$ are bridge poles or they have external degree 1 in H_μ (see Fig. 9d). Let u'_1 be the alias vertex of u_1 with respect to H_{μ_1}

and let w_2' be the alias vertex of w_2 with respect to H_{μ_2}. Note that u_1', w_2' are alias vertices of u, w with respect to H_μ. Let $S_{u_1'w_1'}$ and $S_{u_1'w_1''}$ be two spines of H_{μ_1}, and let $S_{u_2'w_2'}$ and $S_{u_2''w_2'}$ be two spines of H_{μ_2} whose edges and vertices belong to either the left or the right face of H_μ. Let f' be the right face of H_μ (w_1' and u_2' belong to f') and let f'' be the left face of H_μ (w_1'' and u_2'' belong to f''). Let $P_{u_1'u_2'}$ be a subpath of $S_{u_1'w_1'}$ from u_1' to u_2' and let $P_{u_1'u_2''}$ be a subpath of $S_{u_1'w_1''}$ from u_1' to u_2''. Observing that face f' and f'' are R-labeled in the adjacency list of $w_1 \equiv u_2$, we have $n(P_{u_1'u_2'}) = n(S_{u_1'w_1'}) - 1$ and $n(P_{u_1'u_2''}) = n(S_{u_1'w_1''}) + 1$. Let $S'_{u_1'w_2'}$ be a path in H_μ composed by $P_{u_1'u_2'}$ and $S_{u_2'w_2'}$; let $S''_{u_1'w_2'}$ be a path in H_μ composed by $P_{u_1'u_2''}$ and $S_{u_2''w_2'}$. Thus, $n(S'_{u_1'w_2'}) = n(P_{u_1'u_2'}) + n(S_{u_2'w_2'})$ and $n(S''_{u_1'w_2'}) = n(P_{u_1'u_2''}) + n(S_{u_2''w_2'})$. Since both $S'_{u_1'w_2'}$ and $S''_{u_1'w_2'}$ are spines of H_μ, σ_{H_μ} can be written as $(n(S'_{u_1'w_2'}) + n(S''_{u_1'w_2'}))/2$.

(b) $u_1 \equiv u$ is a nonbridge pole with external degree 2 in H_μ (see Figs. 9e and 9f). In this case the spirality of H_μ is evaluated by means of two spines. With analogous reasoning as in the above case, we can define the two spines of H_μ starting from the spines of H_{μ_1} and H_{μ_2}. The proof is a simple variation of the one given above. □

4.4. Spirality of a parallel. Let H be an orthogonal graph and T be the decomposition tree of H. We first consider the case of an internal P-node of T, then the special case of the root.

Let μ be an internal P-node of T with children $\mu_1, \ldots, \mu_k$, let $H_\mu \subset H$ be the pertinent orthogonal graph of μ with poles u and w, and let $H_{\mu_i} \subset H_\mu$ be the pertinent orthogonal graph of μ_i $(i = 1, \ldots, k)$. Since the vertices of H have degree at most 4, μ has two or three children. We relate the spirality of H_μ to the spirality of its components by distinguishing the case where μ has three children from the case where μ has two children.

Suppose first that μ has three children. Let e_i be the edge of H_{μ_i} $(i = 1, 2, 3)$ incident on pole u, and let e_{out} be the external edge of u with respect to H_μ. In the following lemma we suppose that edges e_{out}, e_3, e_2, and e_1 appear in this order in the adjacency list of u. See, for example, Fig. 10.

LEMMA 4.3. *The spirality σ_{H_μ} of H_μ is related to the spiralities $\sigma_{H_{\mu_i}}$ of H_{μ_i} $(i = 1, 2, 3)$ by*

$$\sigma_{H_\mu} = \sigma_{H_{\mu_1}} + 2 = \sigma_{H_{\mu_2}} = \sigma_{H_{\mu_3}} - 2.$$

Proof. Poles u, w are nonbridge poles with external degree 1 in H_μ and bridge poles in H_{μ_i}. Let u', w' be the alias vertices of poles u, w with respect to H_μ, respectively. Let S_2 be a spine of H_{μ_2}; consider the path S composed by edge (u', u), S_2, and edge (w, w'). Observe that S is a spine of H_μ. Since, when going along S from u' to w', we go straight at vertex u and at vertex w, $n(S) = n(S_2)$. Let S_1 be a spine of H_{μ_1}; consider the path S' composed by edge (u', u), S_1, and edge (w, w'); clearly, S' is a spine of H_μ. Since, when going along S' from u' to w', we turn right at vertex u and at vertex w, $n(S') = n(S_1) + 2$. Analogous reasoning applies when a spine of H_μ is defined starting from a spine of H_{μ_3}. □

Consider now that the case μ has two children. Let f_1 and f_2 be the right face and the left face of H_μ, respectively. Suppose that the vertices and edges of H_μ belonging to f_1 are also vertices and edges of H_{μ_1}. We denote with α_v^i the number of R-symbols

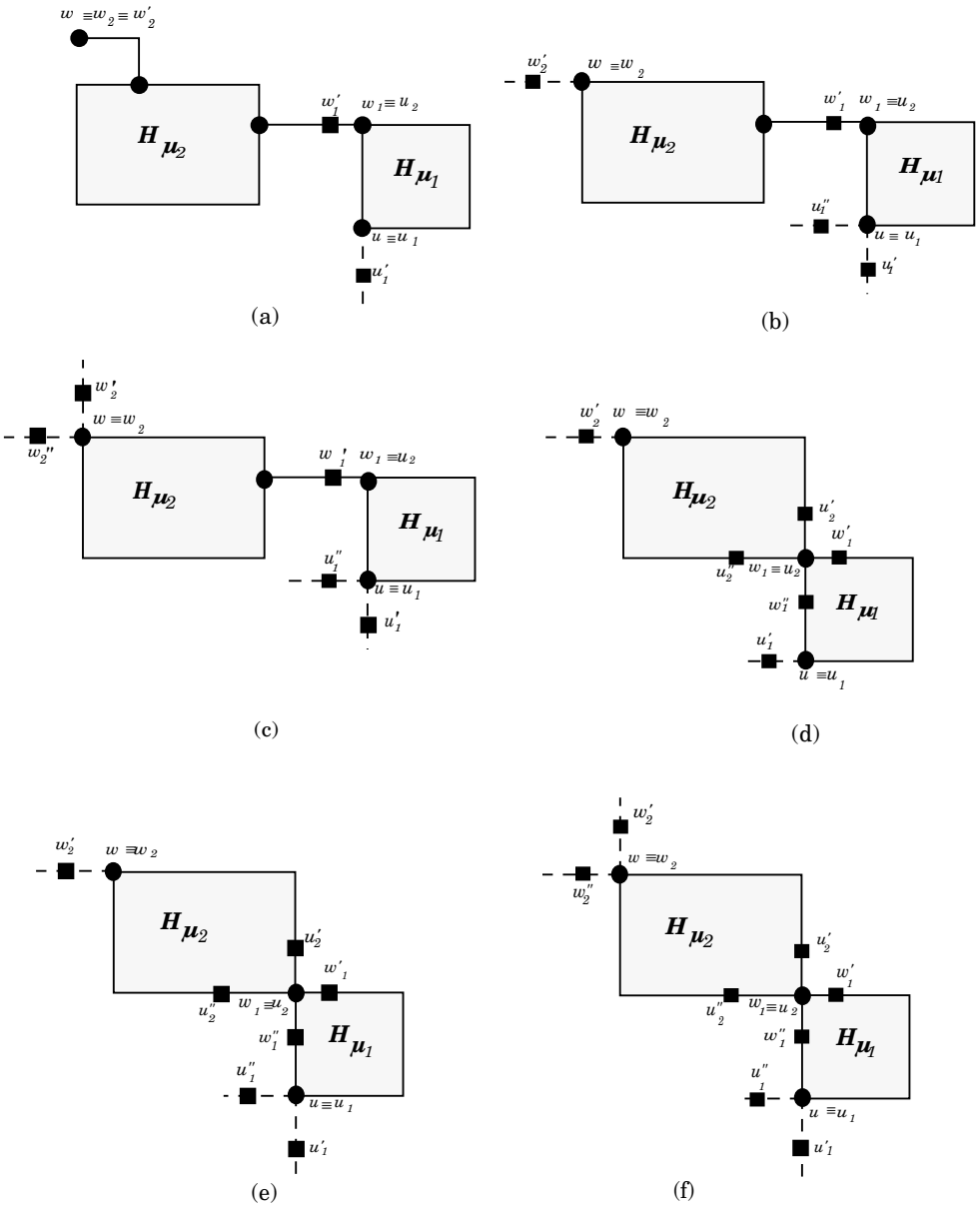


FIG. 9. *Different cases in the proof of Lemma* 4.2.

minus the number of L-symbols of the label associated with face f_i $(i = 1, 2)$ in the adjacency list of pole v $(v = u, w)$. For example, in Fig. 11 we have $\alpha_u^1 = 1$, $\alpha_u^2 = 1$, $\alpha_w^1 = 0$, and $\alpha_w^2 = 1$. Also, we denote with $\text{extdeg}_\mu(v)$ the external degree of v with respect to the pertinent graph of μ and with $\text{intdeg}_\mu(v)$ the internal degree of v with respect to the pertinent graph of μ. The proof of the following lemma can be found in [4].

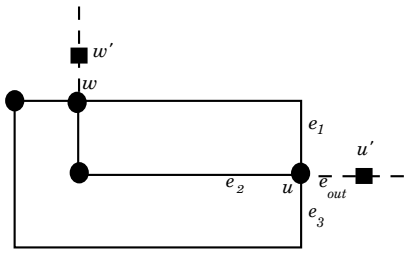


FIG. 10. *Example of a parallel of three components.*

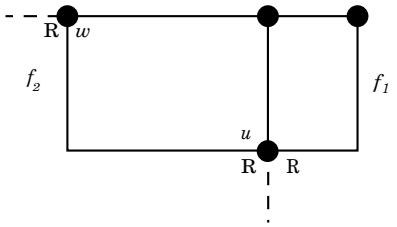


FIG. 11. *Example of a parallel of two components.*

LEMMA 4.4. *The spirality σ_{H_μ} of H_μ is related to the spiralities $\sigma_{H_{\mu_i}}$ of the H_{μ_i} $(i = 1, 2)$ by*

$$\sigma_{H_\mu} = \sigma_{H_{\mu_1}} + k_u \cdot \alpha_u^1 + k_w \cdot \alpha_w^1,$$

$$\sigma_{H_\mu} = \sigma_{H_{\mu_2}} - k_u \cdot \alpha_u^2 - k_w \cdot \alpha_w^2,$$

$$\textit{where} \quad k_v(v = u, w) = \begin{cases} 1 & \textit{if } \mathrm{extdeg}_\mu(v) = 1 \textit{ and } \mathrm{intdeg}_{\mu_i}(v) = 1, \\ 1/2 & \textit{if } \mathrm{extdeg}_\mu(v) = 2 \textit{ or } \mathrm{extdeg}_{\mu_i}(v) = 2. \end{cases}$$

Let μ be the root of T, and let μ_1 and μ_2 be the children of μ such that the pertinent graph of μ_2 is the reference edge (s, t). Let $H_{\mu_1} \subset H$ be the orthogonal pertinent graph of μ_1 and let $H_{st} \subset H$ be the orthogonal representation of edge (s, t). Let f be the internal face of H containing (s, t) and let α_v be the number of R-symbols minus the number of L-symbols of the label associated to f in the adjacency list of pole v $(v = s, t)$. The proof of the following lemma can be found in [4].

LEMMA 4.5. *The spirality $\sigma_{H_{\mu_1}}$ of H_{μ_1} and the spirality $\sigma_{H_{st}}$ of H_{st} are related by*

$$\sigma_{H_{\mu_1}} - \sigma_{H_{st}} + k_s \cdot \alpha_s + k_t \cdot \alpha_t = 4$$

if edge (s, t) is traversed from t to s, going around f in the positive direction, and by

$$-\sigma_{H_{\mu_1}} + \sigma_{H_{st}} + k_s \cdot \alpha_s + k_t \cdot \alpha_t = 4$$

if edge (s, t) is traversed from s to t, going around f in the positive direction, where

$$k_s \ (k_t) = \begin{cases} 1 & \textit{if } s(t) \textit{ is a bridge pole in } H_{\mu_1}, \\ 0 & \textit{otherwise.} \end{cases}$$

4.5. Substituting orthogonal representations with the same spirality. Let G and G' be two embedded graphs describing two different embeddings of the same graph. If they are different only in the embedding of one split component they are said to be *split-different*. More precisely, let $G_{uw} \subset G$ and $G'_{uw} \subset G'$ be two different embedded subgraphs describing two different embeddings of the same split component. For each vertex v that is not in the split component, the clockwise ordering of the edges in the adjacency list of v is the same in G and in G'. For each vertex that is in the split component except poles u and w, the clockwise ordering of the incident edges may be different according to the different embeddings of G_{uw} and G'_{uw}. For poles u and w the clockwise ordering of the external edges is the same in G and G', while the clockwise ordering of the internal edges may be different according to the embedding of G_{uw} and G'_{uw}. Observe that the external faces of G and G' may be different.

Let H and H' be two orthogonal graphs that are orthogonal representations of G and G', respectively, and let $H_{uw} \subset H$ and $H'_{uw} \subset H'$ be the orthogonal graphs that are orthogonal representations of G_{uw} and G'_{uw}. Suppose that the labels that (possibly) appear between the external edges of u and w stay the same in H and in H'. Orthogonal graphs H and H' are said to be *split-different*. Two orthogonal drawings of two split-different orthogonal graphs are shown in Figs. 12a and 12b; they are split-different for the split component in evidence.

Let H and H' be two split-different orthogonal graphs that are orthogonal representations of the same graph G. Suppose they are split-different in a split component with poles u and w, such that $\deg(u) \geq 3$ and $\deg(w) \geq 3$. Let H_{uw} and H'_{uw} be the orthogonal representations of such a split component in H and in H', respectively. Suppose $\sigma_{H_{uw}} = \sigma_{H'_{uw}}$. The operation of *substitution* in H of subgraph H_{uw} with subgraph H'_{uw} gives as a result a labeled embedded graph H'', defined as follows.

1. H'' has the same vertices and edges of G.
2. Each vertex of H'' that is not in the above split component has the same adjacency list it has in H.
3. If u (w) is a bridge pole, then it has in H'' the adjacency list it has in H.
4. If u (w) is a nonbridge pole, then it has in H'' the adjacency list it has in H'.
5. Any other vertex of H'' has the same adjacency list it has in H'.

Roughly speaking, the operation of substitution consists of replacing in H subgraph H_{uw} with H'_{uw}. As an example, if we substitute the split component with poles 4 and 15 of Fig. 12a with the one of Fig. 12b, we obtain the orthogonal graph of Fig. 12c. The following theorem proves that the obtained graph is orthogonal.

THEOREM 4.1. *Let H and H' be two orthogonal 2-connected graphs that are split-different in a split component with poles u and w, such that* $\deg(u) \geq 3$ *and* $\deg(w) \geq 3$. *Let H_{uw} and H'_{uw} be such a split component in H and in H', respectively. Suppose $\sigma_{H_{uw}} = \sigma_{H'_{uw}}$. Let H'' be a labeled embedded graph obtained by substituting in H subgraph H_{uw} with subgraph H'_{uw}. H'' is an orthogonal graph.*

Proof. We denote by H''_{uw} the split component of H'' involved in the operation of substitution. In order to show that H'' is an orthogonal graph, we have to prove that conditions of Property 2.3 are satisfied. Condition 1 of Property 2.3 is trivially satisfied by all vertices of H'' (by the definition of substitution, vertices have the same adjacency list they have in either H or H'). Condition 2 of Property 2.3 is trivially true for each internal face of H''_{uw} and for each face of H'' that is not the left or the right face of H''_{uw}. We show that the condition holds for the right face; the proof for the left face is analogous.

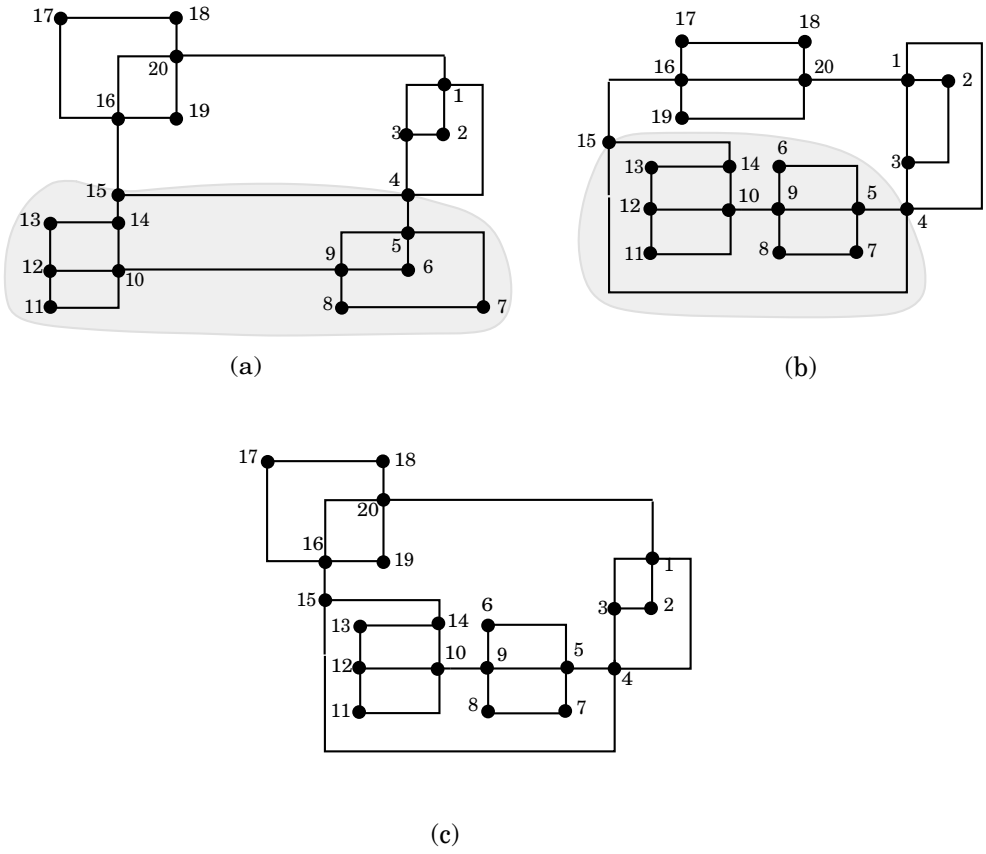


FIG. 12. *An example of substitution.*

Let f_H and $f_{H''}$ be the right faces of H_{uw} and H''_{uw}, respectively. Let u'_H, w'_H be the alias vertices of poles u, w in H that belong to face f_H. Vertices u'_H and w'_H identify two distinct paths S_H and P_H in f_H. Path S_H from u'_H to w'_H is composed by edge (u'_H, u), the right path of H_{uw}, and edge (w, w'_H); path P_H is the other path from w'_H to u'_H. Let $u'_{H''}, w'_{H''}$ be the alias vertices of poles u, w in H'' that belong to face $f_{H''}$. Analogously, $u'_{H''}$ and $w'_{H''}$ identify two distinct paths $S_{H''}$ and $P_{H''}$ in face $f_{H''}$. Observe that $S_{H''}$ is a spine of H''_{uw} in H'', S_H is a spine of H_{uw} in H, and $P_{H''}$ coincides with P_H. By applying Property 2.3 to H, we have $n(S_H)+n(P_H)+a = \pm 4$, where a is the number of R-symbols minus the number of L-symbols of the label associated with face f_H in the adjacency lists of u'_H and w'_H. The plus sign holds if f_H is an internal face, the minus sign, if f_H is the external face.

Consider $f_{H''}$. Since both S_H and $S_{H''}$ have their endpoints on the right faces of their split components, and since H_{uw} and H''_{uw} have the same spirality by hypothesis, $n(S_H) = n(S_{H''})$. Thus, since $n(P_H) = n(P_{H''})$ and the value of a stays the same after the substitution, Condition 2 of Property 2.3 holds for $f_{H''}$. □

4.6. An upper bound to the spirality. We study now the properties of split components of optimal orthogonal representations.

Let G be a graph and let $G_{uw} \subset G$ be a split component of G. Let H be an orthogonal representation of G and let $H_{uw} \subset H$ be the orthogonal representation of

G_{uw}. First, we observe that the orthogonal subgraph H_{uw} may not be optimal even if H is optimal. In Fig. 13 two orthogonal representations of the same graph are shown with a subgraph H_{uw} in evidence; in Fig. 13a the orthogonal representation H is optimal while H_{uw} is not optimal; Fig. 13b shows the best orthogonal representation that can be found if H_{uw} is constrained to be optimal.

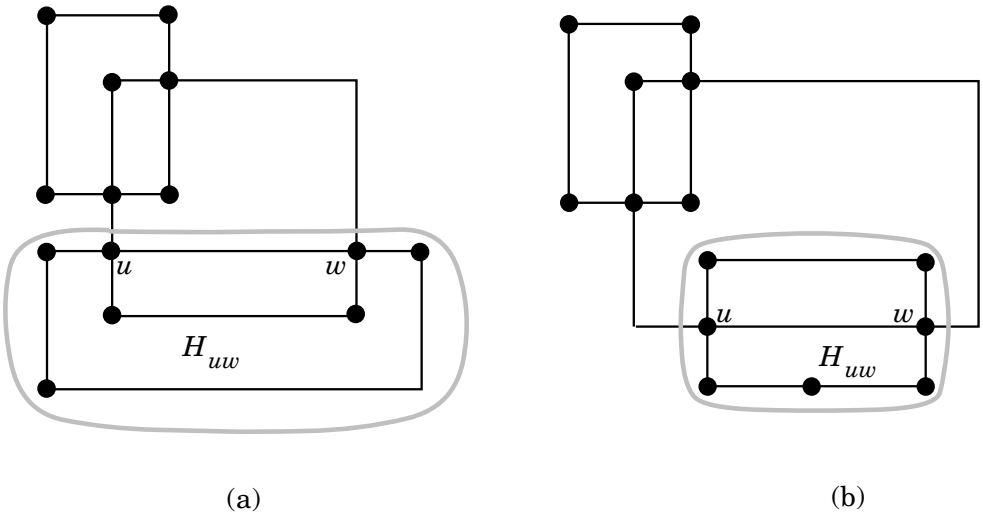


FIG. 13. (a) *H is optimal and H_{uw} is not optimal*; (b) *H is not optimal and H_{uw} is optimal.*

Let H' and H'' be two orthogonal graphs, such that H' and H'' are split-different in the same split component with poles u and w. Let H'_{uw} and H''_{uw} be such split components in H' and in H'', respectively. The orthogonal graph H'_{uw} is *optimal within the spirality* $\sigma_{H'_{uw}}$ if no H''_{uw} exists with spirality $\sigma_{H'_{uw}}$ and fewer bends than H''_{uw}.

THEOREM 4.2. *Let G be a 2-connected graph and let H be an optimal orthogonal representation of G; each split component of H is optimal within its spirality.*

Proof. Let $H_{uw} \subset H$ be the orthogonal representation of a split component of H. Suppose, for a contradiction, that H_{uw} is not optimal within $\sigma_{H_{uw}}$. Then, there exists an orthogonal representation H' of G such that (1) H and H' are split-different in the above split component; (2) the orthogonal representation $H'_{uw} \subset H'$ of the split component in H' has spirality $\sigma_{H_{uw}}$ and fewer bends than H_{uw}. Thus, by Theorem 4.1, it is possible to substitute in H subgraph H_{uw} with H'_{uw} obtaining an orthogonal representation of G with fewer bends than H. This is a contradiction. □

With analogous reasoning we can prove the following theorem.

THEOREM 4.3. *Let G be a 2-connected graph and let $G_{uw} \subset G$ be a split component of G. Let H_{uw} be an orthogonal representation of G_{uw} optimal within spirality $\sigma_{H_{uw}}$. All the split components of H_{uw} are optimal within their spiralities.*

The next theorem gives an upper bound to the spirality of the split components in an optimal orthogonal representation.

THEOREM 4.4. *Let G be a 2-connected graph with n vertices and let H be an optimal orthogonal representation of G; let $H_{uw} \subset H$ be the orthogonal representation of a split component of G, $|\sigma_{H_\mu}| \leq 3n - 2$.*

Proof. Let S be a spine of H_μ. Since S is a simple path, it has at most $n-1$ edges and n vertices. For each pair of consecutive edges of S there is at most one turn at the common endpoint; also, since H is optimal, H has at most $2n-2$ bends [26, 27]. In the worst case, such bends of H are all right turns or they are all left turns along S; thus $|n(S)| \leq 2n-2+n$. □

The split component G_{uw} of G is *odd* if one of its poles is a bridge pole or it has external degree 1, while the other pole is a nonbridge pole with external degree 2 (the total number of alias vertices of u and w with respect to G_{uw} is odd); else G_{uw} is *even* (the total number of alias vertices of u and w with respect to G_{uw} is even). For example, the split component in evidence in Fig. 6 is odd, while the one in evidence in Fig. 13 is even.

Consider now the orthogonal representation $H_{uw} \subset H$ of G_{uw}. Suppose G_{uw} is odd; H_{uw} satisfies case 2 of Definition 4.2 and, according to Property 4.1, its spirality is a fractional number. Otherwise, suppose G_{uw} even; in this case H_{uw} satisfies either case 1 or case 3 of Definition 4.2 and, according to Property 4.1, its spirality is an integer number.

PROPERTY 4.3. *Let k be any integer number. If G_{uw} is odd, there always exists an orthogonal representation H of G such that the orthogonal representation $H_{uw} \subset H$ of G_{uw} has spirality $(2k+1)/2$. If G_{uw} is even, there always exists an orthogonal representation H of G such that the orthogonal representation $H_{uw} \subset H$ of G_{uw} has spirality k.*

Combining Theorem 4.4 and Property 4.3, the following corollary descends.

COROLLARY 4.1. *Let G be a graph with n vertices and let $G_{uw} \subset G$ be a split component of G. Let H be an optimal orthogonal representation of G, and let $H_{uw} \subset H$ be the orthogonal representation of G_{uw}. If G_{uw} is odd, the spirality of H_{uw} is a number $(2k+1)/2$ such that $-3n+2 \leq k \leq 3n-3$. If G_{uw} is even, the spirality of H_{uw} is an integer k such that $-3n+2 \leq k \leq 3n-2$.*

4.7. Spirality and cost functions. The *cost function* of a split component associates to each value σ of spirality the cost (number of bends) of an orthogonal representation of the component optimal within spirality σ.

The following property can be trivially proved.

PROPERTY 4.4. *The cost function of a split component is piecewise linear and it is symmetric with respect to the cost axis (vertical axis corresponding to zero spirality).*

Given an embedding of a split component, a *fixed-embedding cost function* associates with each value σ of spirality the cost of an orthogonal representation with spirality σ, which has the given embedding and the minimum number of bends among all the orthogonal representations of the component with spirality σ and the given embedding.

PROPERTY 4.5. *The cost function of a split component is the lower envelope of all the fixed-embedding cost functions of the component.*

We show that the cost functions of the components of 4-planar graphs can have quite complicated behavior. Namely, they are, in general, neither monotone nor convex. Even for a series-parallel graph a cost function can have a linear number of local minima. This closes the possibility of devising polynomial time algorithms for 4-planar series-parallel graphs based on convexity properties of the cost functions.

Intuitively, one would expect that the number of bends of a split component monotonically increases when the spirality of such a component is augmented. Surprisingly, this is not the behavior for general 4-planar graphs. To give an example,

in Fig. 14 we show the cost function of a series composition that has two minima for $\sigma = 0$ and for $\sigma = 2$. The following lemma shows that the behavior of a cost function can be even worse.

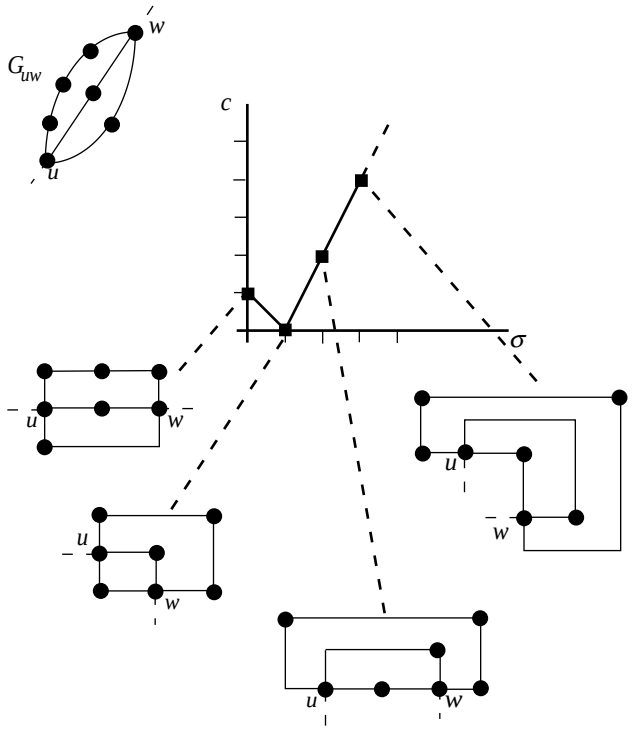


FIG. 14. *An example of nonmonotone cost function.*

LEMMA 4.6. *There exists a split component of a series-parallel graph whose cost function is neither convex nor monotone.*

Proof. Consider the split component of Fig. 15. □

However, the cost function of the split component of Fig. 15a has only one nonconvexity. Now we show an infinite family of split components constructed by only using series and parallel compositions whose cost functions ripple a linear number of times. Let G_n be the split component recursively defined as follows: G_1 is the split component of Lemma 4.6; G_n $(n > 1)$ is the series composition of G_1, one edge, and G_{n-1} (see Fig. 15b). Observe that G_n has $14n$ vertices.

LEMMA 4.7. *The cost function of G_n is zero for $\sigma = \pm 2k$ $(k = 0, \ldots, n)$ and is greater than zero for all the remaining values of σ.*

Proof. For the symmetry of the cost functions (see Property 4.4) we can restrict our attention to nonnegative values of spirality. The proof is by induction on n. We prove first that the cost function of G_1 has value 0 only for spirality 0 and 2.

Let G_0 be the split component of Fig. 14. The split component G_1 is the series composition of two copies of G_0 separated by one edge e_0. The cost function of e_0 is a linear function with slope 1. The cost function of G_0 has value 0 only for spirality 1. Namely, for every value of spirality less than 4, the cost function has the behavior depicted in Fig. 14. Since any orthogonal representation of G_0 has a spine S with three vertices, for values of spirality greater than or equal to 4, such an orthogonal

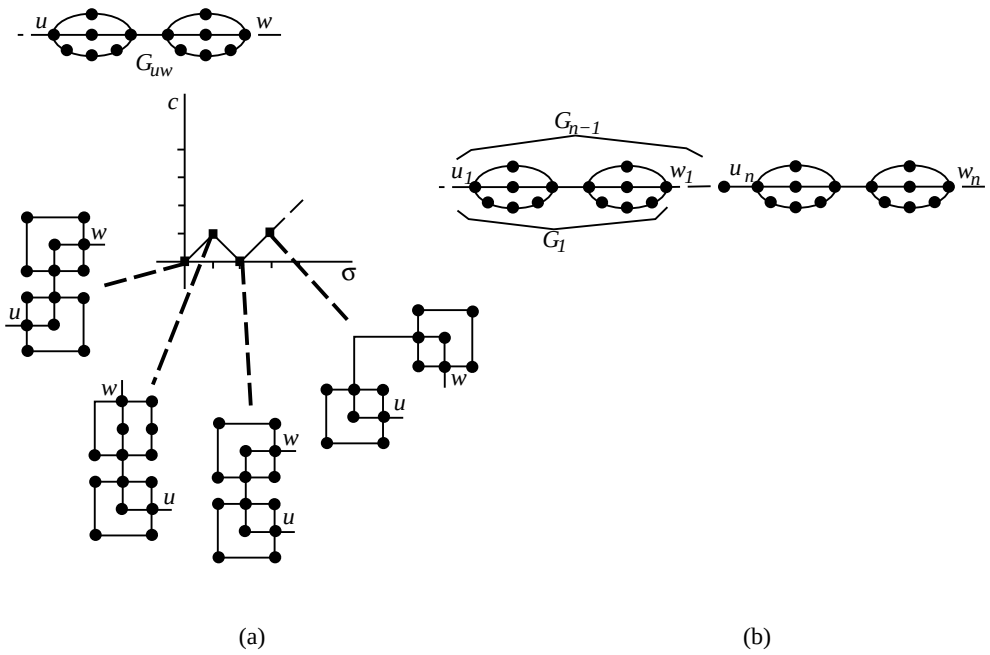


FIG. 15. (a) *An example of nonconvex cost function*; (b) *the series composition* G_n.

representation has at least one bend on S. From Lemma 4.2, the spirality of an orthogonal representation of G_1 is the sum of the spiralities of its components. It follows that an orthogonal representation of G_1 has at least one bend except when the spirality of e_0 is 0 and the spiralities of the orthogonal representations of G_0 are either 1 or -1. Thus, the cost function of G_1 has value 0 only for spirality 0 and 2.

Suppose now the lemma holds for G_{n-1}. We prove the lemma for G_n. G_n is the series of G_{n-1}, one edge e, and G_1. Again, from Lemma 4.2, the spirality of an orthogonal representation of G_n is the sum of the spiralities of its components. It follows that an orthogonal representation of G_n has at least one bend except when the spirality of e is 0 and the spiralities of the orthogonal representations of G_{n-1} and G_1 are such that the corresponding cost functions have value 0. Thus the cost function of G_n has value 0 only for $\sigma = 2k$ $(k = 0, \ldots, n)$. □

The following theorem summarizes the results of this section.

THEOREM 4.5. *There exists an infinite family of* 4*-planar split components constructed by using only series and parallel compositions whose cost functions are neither convex nor monotone.*

5. Cost functions of 3-planar graphs. The results of this section are the basis of a polynomial time algorithm for optimal orthogonal drawings of 3-planar graphs. Namely, let μ be a node of the decomposition tree of a 3-planar graph and let G_μ be its pertinent graph. In section 5.1 we show that the cost function of G_μ is always not decreasing. The contribution of section 5.2 is as follows. First, if μ is a Q^*-node we show that the cost function of G_μ is convex. Second, if μ is an S-node or a P-node, then we show that the convexity of the cost functions of all the children of μ implies the convexity of the cost function of G_μ. Unfortunately, we are not able to prove an analogous property if μ is an R-node. Third, we give more sophisticated tools that

allow us to deal with this case. Namely, we show that the cost functions of all the children of an R-node are convex.

5.1. Not decreasing cost functions. We start by giving a few definitions that will be of use in the rest of the paper.

Let H be an orthogonal 3-planar graph. Let e be an edge of H and let f_1 and f_2 be the faces sharing e. Edge e is an *access edge* from the *starting face* f_1 to the *target face* f_2 if at least one R-symbol is found when traversing e having f_1 on the left side.

Let v be a vertex of H. Two cases are possible.

1. There are two faces f_1 and f_2 in the adjacency list of v, and either f_1 has no labels or it is labeled with an R-symbol; v is an *access vertex* from starting face f_1 to target face f_2.
2. There are three faces f_1, f_2, and f_3 in the adjacency list of v and face f_1 has no label; v is an *access vertex* from starting face f_1 to target faces f_2 and f_3.

An *access* of H is either an access vertex or an access edge of H. For example, in Fig. 16, v_1, v_2, and e are accesses. The starting face of v_1 is f_3 and its target face is f_4; the starting face of v_2 is f_3 and its target faces are f_1 and f_2; the starting face of e is f_e and its target face is f_1.

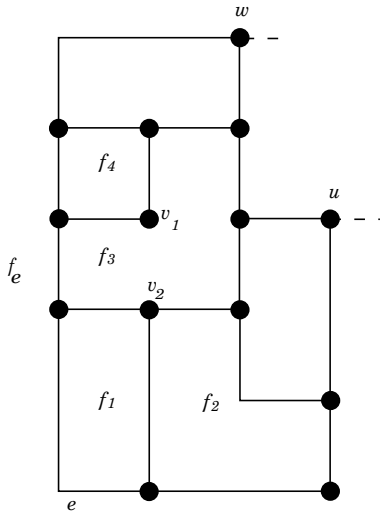


FIG. 16. *An example of accesses.*

Let Γ_1 and Γ_2 be two orthogonal drawings of H, and let C_1 and C_2 be two oriented closed simple curves drawn onto Γ_1 and Γ_2, respectively (curves like C_1 and C_2 have been studied in [21, 26, 27]). Curves C_1 and C_2 are *equivalent curves* if they traverse the same set of edges, vertices, and faces in the same order. In this way the oriented closed simple curves that can be drawn onto the orthogonal drawings of an orthogonal 3-planar graph H are grouped into equivalence classes. A class of equivalent curves is described by a circular oriented list of faces, vertices and edges of H, called the *ring* of H.

A *nonincreasing ring* of H is a ring R such that:

- the reference edge (s, t) belongs to R;
- the external face of H belongs to R and appears immediately after (s, t) in R;

- every edge or vertex of R, except (s,t), is an access a of H. For each sublist f_1, a, f_2 of R, a is an access from f_1 to f_2.

An example of a curve described by a nonincreasing ring is given in Fig. 17a.

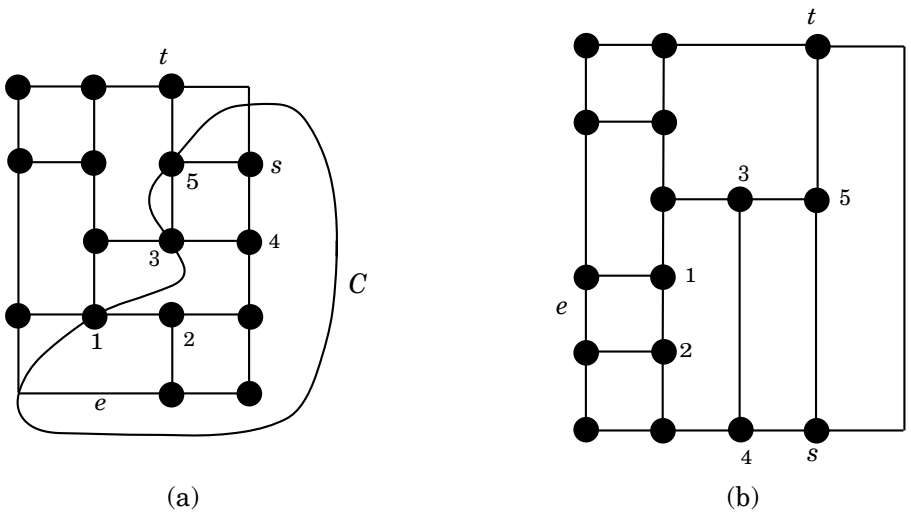


FIG. 17. (a) *A curve C of a nonincreasing ring*; (b) *the elementary transformation defined by C.*

LEMMA 5.1. *Let G be a graph and let G_{st} be the split component of G in parallel with the reference edge (s,t). Let H be an orthogonal representation of G, $H_{st} \subset H$ the orthogonal representation of G_{st}, and R a nonincreasing ring on H. Suppose $\sigma_{H_{st}} > 0$. There exists an orthogonal representation H' of G with the same embedding of H such that $\sigma_{H'_{st}} = \sigma_{H_{st}} - 1$ and H'_{st} has no more bends than H_{st}, where $H'_{st} \subset H'$ is the orthogonal representation of G_{st}.*

Proof. The proof easily follows from a result in [26, 27]. Let Γ be an orthogonal drawing of H. Let C be an oriented closed simple curve drawn onto Γ and such that: (i) C belongs to the class of equivalent curves represented by R; (ii) the intersection between C and (s,t) is not on a bend; and (iii) any other crossing between an edge e of Γ and C is on a bend and is such that C enters from the face where the bend forms a concave angle (this can always be done because e is an access). Curve C defines an *elementary transformation* of the type described in [26, 27]. Such a transformation can be used to modify Γ into a new orthogonal drawing Γ' with one more bend on edge (s,t) (one more left turn when going along (s,t) from s to t) and fewer or as many bends in the rest of the drawing. An example of elementary transformation is shown in Fig. 17, where Figs. 17a and 17b show a drawing before and after the transformation, respectively. In this case the portion of drawing inside C is "rotated" 90 degrees clockwise. Namely, e is straightened, 2 is moved below 1, 4 is moved below 3, 3 is moved to the left of 5, and (s,t) has one more bend. Let H' be the orthogonal graph describing Γ'. Since (s,t) has in H' one more left turn than in H, from Lemma 4.5, the spirality of H'_{st} is equal to $\sigma_{H_{st}} - 1$. □

We are now in a position to prove the following lemma.

LEMMA 5.2. *The cost function of a split component of a 3-planar graph is not decreasing for nonnegative values of spirality.*

Proof. Let T be a decomposition tree of a 3-planar graph and let μ be an internal node of T. Let H'_μ and H''_μ be two orthogonal representations of the pertinent graph

of μ such that H'_μ is optimal within the spirality $\sigma' > 0$ and H''_μ is optimal within the spirality $\sigma'' = \sigma' - 1$. To prove the lemma we need to show that the number of bends of H''_μ is less than or equal to the number of bends of H'_μ.

We first observe that it is sufficient to prove the lemma in a fixed embedding setting. In fact, (1) for Property 4.5 the cost function in the variable embedding setting is the lower envelope of all the fixed embedding cost functions, and (2) the lower envelope of not decreasing functions is itself not decreasing.

If μ is a Q^*-node the proof is trivial. If μ is an S-node and the statement of the lemma holds for each component of the series, then the statement also holds for the entire series (it follows from Lemma 4.2). Thus, we can restrict our attention to the case where μ is either a P-node or an R-node; in both these cases the external degree of the poles of μ is 1.

We show that it is always possible to define a nonincreasing ring for every split component of an orthogonal representation of a 3-planar graph, optimal within a given positive value of spirality. According to Lemma 5.1, the nonincreasing ring allows us to decrease the spirality of the component without increasing its cost. To do that we give a constructive proof. Namely, we give a procedure to compute a nonincreasing ring, and we prove the correctness of such a procedure.

Let H^* be the orthogonal representation obtained by adding edge (u, w) to H'_μ in such a way that (u, w) is the right path of H^* and the spirality of $H'_\mu \subset H^*$ is σ'. Let f_{uw} be the face sharing edge (u, w) with the external face f_e of H^*. Let u' and w' be the alias vertices of u and w, respectively.

Procedure ComputeRing

Step 1: Labeling

Let S be the spine of $H_{\mu'}$ containing the left path of H^*.

repeat

- Let a be the first access of S such that the starting face f_1 is on the left side of S, when going along S from u' to w'.
- Let f_1 be the starting face of a.
- Select a face f_2 according to the following rules: (1) **if** a is an access edge or an access vertex of degree 2, **then** f_2 is the only target face of a; (2) **if** a is an access vertex of degree 3, let $e = (v', a)$ be the edge of S traversed from v' to a; **then** f_2 is the target face of a that contains e (see, for example, Fig. 18a).
- Label access a with the ordered pair f_1, f_2.
- Let v_1 (v_2) be the first (last) encountered vertex of f_2 when going along S from u' to w'. Let P_S be the subpath of S with endpoints v_1 and v_2 and let P be the path joining v_1 and v_2 along f_2 such that P shares with S only its endpoints.
- Replace path P_S of S with path P. (See, for example, Fig. 18b.)

until a is an access with target face f_{uw}

Label access a with the ordered pair f_1, f_{uw}, where f_1 is the starting face of a; also, label edge (u, w) with f_{uw}, f_e.

Step 2: Construction

Construct the ring R containing $f_{u,w}$ such that for each f_1, a, f_2 of R, access a has been labeled f_1, f_2 in Step 1.

end procedure.

Procedure ComputeRing correctly constructs a nonincreasing ring. Observe first that any spine S of $H_{\mu'}$ has at least one access, because σ' is positive and S has at least one right turn; also, there exists at least one spine with an access having target face f_{uw} (consider, as an example of such a spine, the spine containing the right path of $H_{\mu'}$).

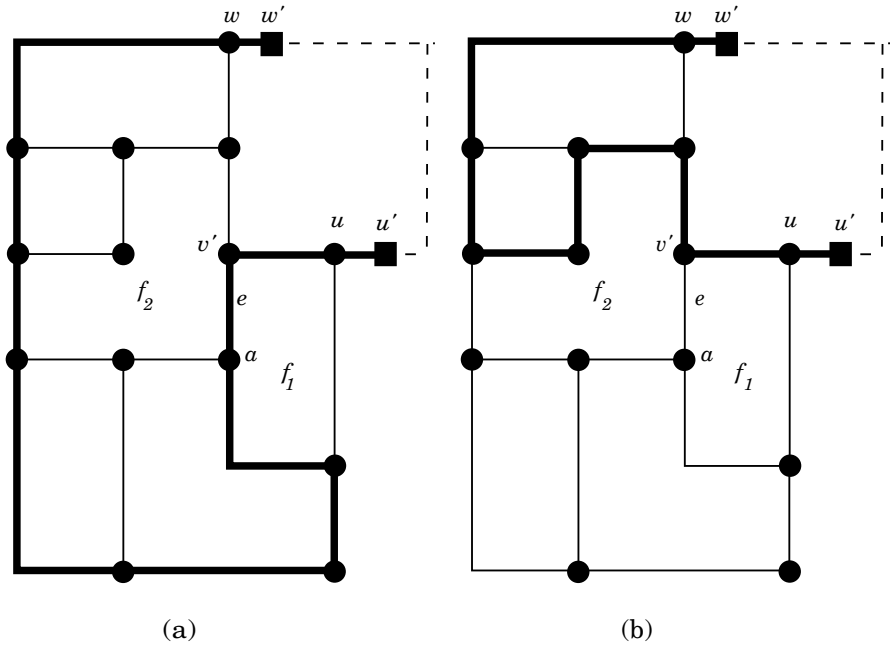


FIG. 18. *An example of choice of the access in the proof of Lemma* 5.2.

Consider now the repeat cycle of the labeling step (Step 1) of the procedure. At each iteration a new spine is computed, by replacing a subpath P_S of the current spine S with a path P sharing with S only its endpoints. Namely, let P_1 be the subpath of S joining u' to v_1 and P_2 be the subpath of S from v_2 to w'. Paths P_1 and P_2 are simple and disjoint by definition. Path P computed in the while cycle is a simple path that intersects P_1 only in v_1 and P_2 only in v_2. Thus, the concatenation of P_1, P, and P_2 is a spine.

Observe that the number of faces between the spine and the left path of H^* increases at each iteration. Intuitively, the spine sweeps from the left path of H^* towards the right path by *eating* some faces on the right of S (between S and the right path of H^*). No face can be eaten twice, since an eaten face moves from the right to the left of the current spine.

An access with target face f_{uw} is found after a finite number of iterations because the cardinality of the set of faces on the right of the current spine decreases at each iteration.

Concerning the construction step (Step 2) of the procedure, we have that an increasing ring R exists. In fact, if R contains face f, then there is a labeled access a having f as target face and labeled with the pair f_1, f. But, since a is a labeled access, a belongs to a spine S having the starting face f_1 at its left. Now, either face f_1 is the face f_e and we are done, or the spine S has been obtained from a previous

spine by eating face f_1. This means that f_1 is the target face of some previously labeled access and we can repeat the argument. □

5.2. Convexity properties of the cost functions. A cost function $c(\sigma)$ of a split component is an *elbow-shaped function* when there exists a pair $\overline{\sigma}$, k such that $c(\sigma) = k$ for $-\overline{\sigma} \leq \sigma \leq \overline{\sigma}$ and $c(\sigma) = |\sigma| + k - \overline{\sigma}$ for $\overline{\sigma} \leq \sigma$.

THEOREM 5.1. *Let G be a 3-planar 2-connected graph and let T be its decomposition tree. Let μ be a node of T and let G_μ be its pertinent graph.*

1. *If μ is a Q^*-node or an S-node, then the cost function of G_μ is elbow-shaped.*
2. *If μ is a P-node, then the cost function of G_μ is convex.*
3. *If μ is an R-node, then the cost functions of the pertinent graphs of the children of μ are elbow-shaped.*

The proof of Theorem 5.1 is based on inductive arguments and will be given in section 5.2.3. The following Lemmas 5.3, 5.4, 5.5, and 5.6 are the main tools of the induction.

LEMMA 5.3. *Let μ be a Q^*-node whose pertinent graph G_μ has m edges. The cost function of G_μ is an elbow-shaped function such that $k = 0$ and $\overline{\sigma} = m - 1$.*

Proof. Suppose that $\sigma \geq 0$; the proof is analogous for $\sigma < 0$. For $\sigma = 0$ an orthogonal drawing of G_μ can be easily constructed by putting all the vertices on the same straight line. For each increase of the spirality of one unit, we can bend the drawing at one of the $m - 1$ internal vertices that did not bend before, without introducing any extra cost until $\sigma = m - 1$. For $\sigma > m - 1$ we need to add $\sigma - m + 1$ bends, and this can be done along a distinguished edge. □

LEMMA 5.4. *Let μ be an S-node whose pertinent graph is G_μ; let μ_1 and μ_2 be the children of μ and let G_{μ_1} and G_{μ_2} be the pertinent graphs of μ_1 and μ_2, respectively. If the cost function of G_{μ_1} is convex and the cost function of G_{μ_2} is elbow-shaped, then the cost function of G_μ is elbow-shaped.*

Proof. Let $c(\sigma)$, $c_1(\sigma)$, and $c_2(\sigma)$ be the cost functions of G_μ, G_{μ_1}, and G_{μ_2}, respectively. For Property 4.4, it is sufficient to show that $c(\sigma)$ is elbow-shaped for nonnegative values of spirality.

Since, for Lemma 5.2, $c_1(\sigma)$ is not decreasing when $\sigma \geq 0$ and since $c_2(\sigma)$ is elbow-shaped, an increase of one unit of spirality for the whole series causes an increase of at most one unit on the overall number of bends. In fact, one can conveniently increase by one unit the spirality on the whole series by bending the component for which the increase on the cost function is the least expensive. Observe that $c(0) = c_1(0) + c_2(0)$. Let σ_1 and σ_2 be the maximum values of spirality for which $c_1(\sigma) = c_1(0)$ and $c_2(\sigma) = c_2(0)$, respectively. Now, because of Lemma 4.2, one can have no cost increase for all values of spirality $0 \leq \sigma \leq \sigma_1 + \sigma_2$. For $\sigma > \sigma_1 + \sigma_2$, for the convexity of $c_1(\sigma)$ and $c_2(\sigma)$, $c(\sigma)$ has a positive slope. A unit slope for $c(\sigma)$ can be obtained by fixing $c(\sigma) = c_1(\sigma_1) + c_2(\sigma - \sigma_1)$. □

LEMMA 5.5. *Let μ be a P-node whose pertinent graph is G_μ; let μ_1 and μ_2 be the children of μ and let G_{μ_1} and G_{μ_2} be the pertinent graphs of μ_1 and μ_2, respectively. If the cost functions of G_{μ_1} and G_{μ_2} are elbow-shaped, then the cost function of G_μ is convex.*

Proof. Let $c(\sigma)$, $c_1(\sigma)$, and $c_2(\sigma)$ be the cost functions of G_μ, G_{μ_1}, and G_{μ_2}, respectively. Consider any triplet of consecutive, nonnegative values of σ, say $\overline{\sigma} - 1$, $\overline{\sigma}$, and $\overline{\sigma} + 1$. Let H_μ, H'_μ, and H''_μ be the orthogonal representations of G_μ optimal within spiralities $\overline{\sigma} - 1$, $\overline{\sigma}$, and $\overline{\sigma} + 1$, respectively.

For Lemma 5.2 the cost function $c(\sigma)$ is not decreasing for $\sigma \geq 0$. Also, for Property 4.4, $c(\sigma)$ is symmetric with respect to the y-axis. Thus, to prove the convexity

of $c(\sigma)$ it is sufficient to show that

$$c(\overline{\sigma}+1) - c(\overline{\sigma}) \geq c(\overline{\sigma}) - c(\overline{\sigma}-1).$$

We distinguish two cases.

1. The relative position of the two components of the parallel does not change in H_μ, H'_μ, and H''_μ (i.e., the right path of the parallel belongs to the same split component in H_μ, H'_μ, and H''_μ). Suppose, w.l.o.g., that the right path of G_μ belongs to G_{μ_1}.

   From Lemma 4.4, we have

   $$c(\overline{\sigma}) = \min(c_1(\overline{\sigma} - \alpha_u^1 - \alpha_w^1) + c_2(\overline{\sigma} + \alpha_u^2 + \alpha_w^2)),$$

   where the minimum is over all the possible values of the parameters α_u^1, α_w^1, α_u^2, α_w^2. Since $c_1(\sigma)$ and $c_2(\sigma)$ are elbow-shaped, if $\overline{\sigma} - 1 \geq 2$, the above minimum gives rise to the following equalities, where $\alpha_u^1 = \alpha_w^1 = 1$ and $\alpha_u^2 = \alpha_w^2 = 0$:

   $$c(\overline{\sigma}-1) = c_1(\overline{\sigma}-3) + c_2(\overline{\sigma}-1),$$

   $$c(\overline{\sigma}) = c_1(\overline{\sigma}-2) + c_2(\overline{\sigma}),$$

   $$c(\overline{\sigma}+1) = c_1(\overline{\sigma}-1) + c_2(\overline{\sigma}+1).$$

   If $\overline{\sigma} - 1 < 2$, the values of parameters α_u^1, α_w^1, α_u^2, α_w^2 that minimize the above formulas might change, but the reasoning of the proof is the same. We prove the lemma for $\overline{\sigma} - 1 \geq 2$.

   By using the above equations we derive

   $$c(\overline{\sigma}+1) - c(\overline{\sigma}) = c_1(\overline{\sigma}-1) + c_2(\overline{\sigma}+1) - c_1(\overline{\sigma}-2) - c_2(\overline{\sigma})$$

   and

   $$c(\overline{\sigma}) - c(\overline{\sigma}-1) = c_1(\overline{\sigma}-2) + c_2(\overline{\sigma}) - c_1(\overline{\sigma}-3) - c_2(\overline{\sigma}-1).$$

   Because of the elbow-shape of $c_1(\sigma)$ and $c_2(\sigma)$, we have $c_1(\overline{\sigma}-1) - c_1(\overline{\sigma}-2) \geq c_1(\overline{\sigma}-2) - c_1(\overline{\sigma}-3)$ and $c_2(\overline{\sigma}+1) - c_2(\overline{\sigma}) \geq c_2(\overline{\sigma}) - c_2(\overline{\sigma}-1)$. The thesis easily follows.

2. The relative position of the two components of the parallel changes in H_μ, H'_μ, and H''_μ.

   Let $c_A(\sigma)$ and $c_B(\sigma)$ be the two fixed-embedding cost functions obtained by considering the two possible relative positions of G_{μ_1} and G_{μ_2}. Functions $c_A(\sigma)$ and $c_B(\sigma)$ are clearly convex from what was given above. Since the relative position of the components of the parallel changes with the spirality, $c_A(\sigma)$ and $c_B(\sigma)$ cross between $\overline{\sigma}-1$ and $\overline{\sigma}+1$. We show that this is not the case. Again, we restrict our attention to the case $\overline{\sigma} - 1 \geq 2$. The proof for $\overline{\sigma} - 1 < 2$ is analogous.

Suppose, w.l.o.g., that $c_A(\overline{\sigma}-1) < c_B(\overline{\sigma}-1)$ and $c_A(\overline{\sigma}+1) > c_B(\overline{\sigma}+1)$. Given the considerations stated in case 1, this implies that

$$c_A(\overline{\sigma}-1) = c_1(\overline{\sigma}-3) + c_2(\overline{\sigma}-1),$$

$$c_A(\overline{\sigma}+1) = c_1(\overline{\sigma}-1) + c_2(\overline{\sigma}+1),$$

$$c_B(\overline{\sigma}-1) = c_1(\overline{\sigma}-1) + c_2(\overline{\sigma}-3),$$

$$c_B(\overline{\sigma}+1) = c_1(\overline{\sigma}+1) + c_2(\overline{\sigma}-1).$$

It follows that

$$c_1(\overline{\sigma}+1) - c_1(\overline{\sigma}-1) < c_2(\overline{\sigma}+1) - c_2(\overline{\sigma}-1) \tag{1}$$

and that

$$c_1(\overline{\sigma}-1) - c_1(\overline{\sigma}-3) > c_2(\overline{\sigma}-1) - c_2(\overline{\sigma}-3). \tag{2}$$

Since $c_1(\sigma)$ is an elbow-shaped function, from inequality (2), $1 \leq c_1(\overline{\sigma}-1) - c_1(\overline{\sigma}-3) \leq 2$ that can be rewritten as $1 \leq c_1(\overline{\sigma}-1) - c_1(\overline{\sigma}-2) + c_1(\overline{\sigma}-2) - c_1(\overline{\sigma}-3) \leq 2$; thus, we have that $c_1(\overline{\sigma}-1) - c_1(\overline{\sigma}-2) = 1$, and $c_1(\overline{\sigma}+1) - c_1(\overline{\sigma}-1) = 2$.

But, from inequality (1), $c_1(\overline{\sigma}+1) - c_1(\overline{\sigma}-1) \leq 1$. This is a contradiction. □

To deal with R-nodes we need more sophisticated tools and introduce the concept of *fixed-ordering cost function*, defined as follows. Given an R-node μ with pertinent graph G_μ of a 3-planar graph, with poles u and w, the possible orderings for the edges of G_μ around u and w are exactly two; one can be obtained from the other by flipping the component around the poles. Call such orderings *ordering A* and *ordering B*. Suppose ordering A is given. For each value σ of spirality there is an orthogonal representation of the component that has spirality σ, the minimum number of bends, and ordering A. We call such an orthogonal representation *optimal within spirality σ and ordering A*. It is worth noticing that an orthogonal representation optimal within spirality σ and ordering A is computed by considering all the possible embeddings for the components of G_μ; only the ordering of the edges around u and w is fixed. A fixed-ordering cost function describes the cost of orthogonal representations of the pertinent graphs of R-nodes that are optimal within given values of spirality and ordering. Note that a fixed-ordering cost function is, in general, nonsymmetric with respect to the cost axis. In what follows we denote with $c_A(\sigma)$ and with $c_B(\sigma)$ the fixed-ordering cost functions relative to ordering A and ordering B, respectively.

LEMMA 5.6. *Let μ be an R-node whose pertinent graph is G_μ. If the cost functions of the pertinent graphs of the children of μ are elbow-shaped, then the two fixed-ordering cost functions of G_μ are convex.*

The proof of Lemma 5.6 will be given in sections 5.2.1 and 5.2.2. It is based on the study of the fixed-ordering cost function associated to the pertinent graph G_μ, in terms of the analysis of the flow variation in a min-cost flow network. This exploits a technique already introduced by Tamassia [21]. In the next subsection we describe the flow model.

5.2.1. The flow network. Let T be a decomposition tree of a 3-planar graph and let μ be an R-node of T with poles u and w. Let G_μ be the pertinent graph of μ and let the two possible orderings for the edges of G_μ around poles u and w be given.

Let $skeleton_A(\mu)$ ($skeleton_B(\mu)$) be the embedded graph obtained by adding (u, w) to $skeleton(\mu)$ and such that (1) (u, w) is on the external face, (2) (u, w) is the right path, and (3) the edges of $skeleton_A(\mu)$ incident on u and w correspond to split components of G_μ that respect ordering A (ordering B).

We associate to μ, σ, and ordering A a network $N_\mu(\sigma, A)$ defined as follows.

- Nodes:
  - —There is a node (*vertex-node*) for each vertex of $skeleton_A(\mu)$.
  - —There is a node (*face-node*) for each face of $skeleton_A(\mu)$.
  - —There are two extra nodes: a source node s and a sink node t.
- Arcs:
  - —There are two arcs (f', f'') and (f'', f') for each pair of faces sharing at least one edge. The arcs have capacity, lower bound, and cost assigned with the following rules. (i) If faces f' and f'' share (u, w), and f'' is the external face of $skeleton_A(\mu)$, then arcs (f', f'') and (f'', f') have zero cost; if $\sigma \leq 4$ then arc (f', f'') has lower bound and capacity equal to $4 - \sigma$, and arc (f'', f') has zero capacity; if $\sigma > 4$ then arc (f'', f') has lower bound and capacity with value $\sigma - 4$, and arc (f', f'') has zero capacity. (ii) If faces f' and f'' share an edge e, then arcs (f', f'') and (f'', f') have infinite capacity, lower bound zero, and cost defined by the cost function of the corresponding split component of G_μ with the endpoints of e as poles.
  - —There is one arc (v, f) where v is a vertex that belongs to face f. Arc (v, f) has infinite capacity and zero lower bound and cost.
  - —There is one arc (s, f) for each internal face f composed by fewer than four edges. Arc (s, f) has capacity 4 minus the number of edges belonging to the face and zero lower bound and cost.
  - —There is one arc (s, v) for each vertex v. Arc (s, v) has capacity equal to $4 - \deg(v)$ and zero lower bound and cost.
  - —There is one arc (f, t) for each internal face f composed by more than four edges. Arc (f, t) has capacity equal to the number of edges belonging to the face minus 4 and zero lower bound and cost.
  - —There is one arc (f_e, t) for the external face f_e. Arc (f_e, t) has capacity equal to 4 plus the number of edges belonging to the face and zero lower bound and cost.

An example of a network is shown in Fig. 19. In this figure we represent only vertex-nodes, face-nodes, and arcs between adjacent faces. For two arcs we also draw (in the circles) the corresponding cost functions.

Similarly, a network $N_\mu(\sigma, B)$ is associated with μ, σ, and ordering B.

The flow value z in $N_\mu(\sigma, A)$ is constrained equal to the sum of the capacities of the arcs' outgoing node s.

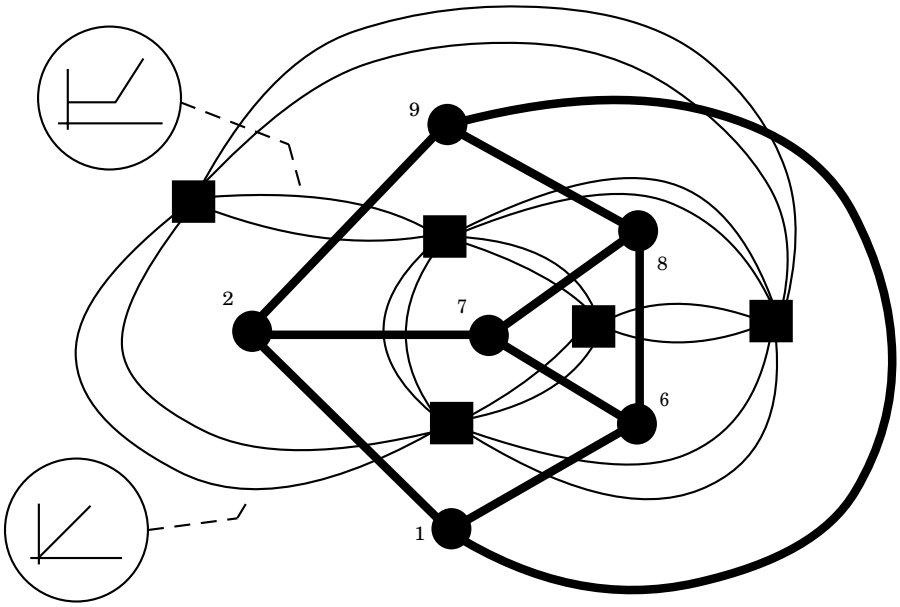


FIG. 19. *Part of the network associated with the skeleton of the R-node of Fig.* 3.

The intuitive interpretation of network $N_\mu(\sigma, A)$ is the following. Each unit of flow in the network represents an angle of 90 degrees; for each pair (f', f'') and (f'', f') linking two face-nodes, their difference of flow represents the spirality of the split components whose virtual edge is the edge separating f' and f'' in $skeleton_A(\mu)$; the cost of the flow represents the number of bends of the orthogonal representation of G_μ (observe that it is not the cost of an orthogonal representation of $skeleton_A(\mu)$). Furthermore, the lower bound of the dual arc of (u, w) constrains edge (u, w) to have the given spirality.

Observe that all the components of 3-planar graphs are even and, by Property 4.3, their possible values of spirality are integer numbers. The proofs of the next two lemmas can be found in [4].

LEMMA 5.7. *Let H_μ be an orthogonal representation of G_μ optimal within spirality σ and ordering A (ordering B) and let c_A (c_B) be the number of bends of H_μ. Then there exists an integer flow in $N_\mu(\sigma, A)$ ($N_\mu(\sigma, B)$) with value z_A (z_B) and cost c_A (c_B).*

LEMMA 5.8. *Let x be a minimum cost integer flow of value z_A (z_B) in $N_\mu(\sigma, A)$ ($N_\mu(\sigma, B)$) and cost $c_A(x)$ ($c_B(x)$). There exists an orthogonal representation H_μ of G_μ optimal within spirality σ and ordering A (ordering B) whose number of bends is $c_A(x)$ ($c_B(x)$).*

The following lemma summarizes the results of this subsection.

LEMMA 5.9. *For each orthogonal representation optimal within spirality σ and ordering A (ordering B) there exists a minimum cost integer flow in $N_\mu(\sigma, A)$ ($N_\mu(\sigma, B)$). Furthermore, an orthogonal representation optimal within spirality σ and ordering A (ordering B) can be computed from the minimum cost integer flow in $N_\mu(\sigma, A)$ ($N_\mu(\sigma, B)$).*

5.2.2. Proof of Lemma 5.6.

Proof. We first show that the fixed-ordering cost functions of the pertinent graph G_μ of μ are convex. Let $c_A(\sigma)$ and $c_B(\sigma)$ be the two fixed-ordering cost functions of G_μ. We prove the convexity of $c_A(\sigma)$; the proof for $c_B(\sigma)$ is analogous. Namely, we show that for each triplet of consecutive values of σ, say $\overline{\sigma}-1$, $\overline{\sigma}$, and $\overline{\sigma}+1$, we have

$$c_A(\overline{\sigma}+1) - c_A(\overline{\sigma}) \geq c_A(\overline{\sigma}) - c_A(\overline{\sigma}-1). \tag{3}$$

Inequality (3), together with Lemma 5.2, proves the convexity of $c_A(\sigma)$.

Let u, w be the poles of G_μ and let f_{uw} be the internal face of $skeleton_A(\mu)$ sharing edge (u, w) with the external face f_e. Let $N_\mu(\overline{\sigma}-1, A)$, $N_\mu(\overline{\sigma}, A)$, and $N_\mu(\overline{\sigma}+1, A)$ be the networks associated with μ for the values of spirality $\overline{\sigma}-1$, $\overline{\sigma}$, $\overline{\sigma}+1$, respectively. Such networks differ only for the values of capacity and lower bound of the pair of arcs between f_e and f_{uw}. For example, consider $N_\mu(\overline{\sigma}-1, A)$ and $N_\mu(\overline{\sigma}, A)$. If $\overline{\sigma}-1 < 4$, then $N_\mu(\overline{\sigma}, A)$ can be obtained from $N_\mu(\overline{\sigma}-1, A)$ by decreasing by one unit the capacity and the lower bound of arc (f_e, f_{uw}); otherwise it can be obtained by increasing by one unit the capacity and the lower bound of (f_{uw}, f_e). We consider the case when the capacity and the lower bound of (f_e, f_{uw}) decrease. The proof is analogous when the capacity and the lower bound of (f_{uw}, f_e) increase.

Let H_μ be an orthogonal representation of G_μ optimal within spirality $\overline{\sigma}-1$ and ordering A; let $\overline{x}$ be the flow in $N_\mu(\overline{\sigma}-1, A)$ that, according to Lemma 5.9, corresponds to H_μ.

We remind the reader that for arc (f_e, f_{uw}) the capacity and the lower bound coincide and that (f_e, f_{uw}) is a saturated arc in $N_\mu(\overline{\sigma}-1, A)$ when the flow is $\overline{x}$. Thus, an optimal flow $\overline{x}'$ in $N_\mu(\overline{\sigma}, A)$ can be computed starting from flow $\overline{x}$ and by decreasing the flow in the arc (f_e, f_{uw}) by means of a minimum cost augmenting path P' connecting f_e and f_{uw} (see also [1]). The objective function (i.e., the cost function $c_A(\sigma)$ of G_μ) changes by an amount $d(P')$, that is, the cost of the chosen augmenting path P' (in the rest of the proof we denote by $d(P)$ the cost of an augmenting path P). Let H'_μ be the orthogonal representation of G_μ computed from $N_\mu(\overline{\sigma}, A)$ and $\overline{x}'$. From Lemma 5.9, H'_μ is optimal within spirality $\overline{\sigma}$ and ordering A. Also, the number of bends in H'_μ is $c_A(\overline{\sigma}-1) + d(P')$.

Consider now network $N_\mu(\overline{\sigma}+1, A)$. Again, an optimal flow $\overline{x}''$ for $N_\mu(\overline{\sigma}+1, A)$ is computed starting from $\overline{x}'$, by means of a minimum cost augmenting path P'' from f_e to f_{uw}. The corresponding orthogonal representation H''_μ is optimal within spirality $\overline{\sigma}+1$ and ordering A and has $c_A(\overline{\sigma}-1) + d(P') + d(P'')$ bends.

It follows that inequality (3) can be rewritten as $d(P'') \geq d(P')$. Suppose, for a contradiction, that $d(P'') < d(P')$. Three cases are possible.

1. P' and P'' do not share common edges. In this case P' is not the minimum cost augmenting path when computing flow $\overline{x}'$ from flow $\overline{x}$. This is a contradiction.
2. P' and P'' share edge $e = (p, q)$ traversed in the same direction in both P' and P'' (see Fig. 20a). In other words, the flow along e is increased or decreased in both P' and P''. In both cases, since the cost function of e is convex, the variation $d'(e)$ of the cost along e in P' is not greater than the variation $d''(e)$ of the cost along e in P''. Path P' is the concatenation of three subpaths, P'_1, edge e, and P'_2. Analogously, P'' is the concatenation of subpaths P''_1, edge e, and P''_2. Inequality $d(P'') < d(P')$ can be rewritten as follows.

$$d(P''_1) + d''(e) + d(P''_2) < d(P'_1) + d'(e) + d(P'_2). \tag{4}$$

Since path $P'_1 e P'_2$ is chosen as a minimum cost augmenting path in network $N_\mu(\overline{\sigma}, A)$, we have that $d(P')$ is not greater than the cost of the paths $P'_1 e P''_2$ and $P''_1 e P'_2$ (if a path is not an augmenting path its cost is infinite). Thus, the following inequalities also hold:

$$d(P'_1) + d'(e) + d(P'_2) \leq d(P'_1) + d'(e) + d(P''_2),$$

$$d(P'_1) + d'(e) + d(P'_2) \leq d(P''_1) + d'(e) + d(P'_2),$$

which imply

$$d(P'_2) \leq d(P''_2),$$

$$d(P'_1) \leq d(P''_1).$$

The last two inequalities, together with $d''(e) \leq d'(e)$ contradict inequality (4).

3. P' and P'' share edge e, with endpoints p and q, traversed in the opposite direction in P' and in P'' (see Fig. 20b). The variation $d'(e)$ of the cost along e in P' is equal to the variation $-d''(e)$ of the cost along e in P''. P' is the concatenation of three subpaths, P'_1 from f_e to p, edge (p, q), and P'_2 from q to f_{uw}. Analogously, P'' is the concatenation of subpaths P''_1 from f_e to q, edge (q, p), and P''_2 from p to f_{uw}.
Inequality $d(P'') < d(P')$ can be rewritten as follows:

$$d(P''_1) - d'(e) + d(P''_2) < d(P'_1) + d'(e) + d(P'_2). \tag{5}$$

Since path P' is chosen as a minimum cost augmenting path in network $N_\mu(\overline{\sigma}, A)$, we have that $d(P')$ is not greater than the cost of the paths $P'_1 P''_2$ and $P'_2 P''_1$. Thus

$$d(P'_1) + d'(e) + d(P'_2) \leq d(P'_1) + d(P''_2),$$

$$d(P'_1) + d'(e) + d(P'_2) \leq d(P'_2) + d(P''_1),$$

which imply

$$d'(e) + d(P'_2) \leq d(P''_2),$$

$$d(P'_1) + d'(e) \leq d(P''_1).$$

The last two inequalities contradict inequality (5). In fact, we can write

$$d(P_1') + d'(e) - d'(e) + d'(e) + d(P_2') < d(P_1') + d'(e) + d(P_2'),$$

$$d(P_1') + d(P_2') < d(P_1') + d(P_2'). \qquad \square$$

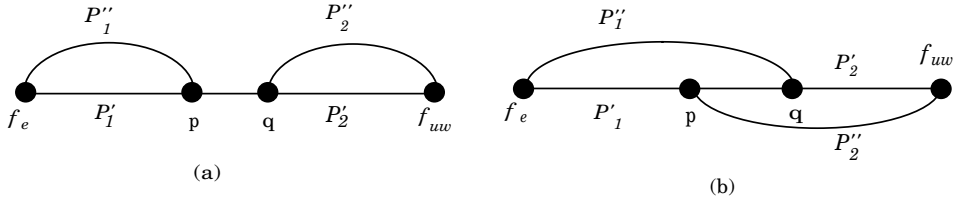


FIG. 20. (a) *Illustration for case* 2 *in the proof of Lemma* 5.6; (b) *illustration for case* 3 *in the proof of Lemma* 5.6.

5.2.3. Proof of Theorem 5.1.

Proof. The proof is by induction on the height of the subtree T_μ rooted at μ.

Base case. If T_μ has height 0, then by the definition of decomposition tree μ is a Q^*-node; the thesis follows by Lemma 5.3. If T_μ has height 1, then by the definition of decomposition tree, the children of μ are Q^*-nodes; the thesis follows from Lemmas 5.3 and 5.5.

Inductive case. Suppose T_μ has height $k > 1$. Three cases are possible: μ is an R-node, a P-node, or an S-node.

If μ is an R-node or a P-node, then by Property 2.2 its children are either S-nodes or Q^*-nodes. Hence, the pertinent graphs of the children of μ have elbow-shaped cost functions because of the inductive hypothesis. If μ is an R-node, then this is enough to prove the theorem. If μ is a P-node, then the thesis follows from the above discussion and from Lemma 5.5.

If μ is an S-node and none of its children is an R-node or a P-node, the theorem follows from the inductive hypothesis and from Lemma 5.4.

If μ is an S-node and one of its children is a P-node, from Property 2.2 the other child of μ can be either an S-node or a Q^*-node. The theorem follows from the inductive hypothesis and from Lemma 5.4.

Suppose now that μ is an S-node with children μ_1 and μ_2, and one of its children, say μ_1, is an R-node. Because of Property 2.2, the children of μ_1 are either S-nodes or Q^*-nodes. So, for the inductive hypothesis on the children of μ_1 and for Lemma 5.6, both the fixed-ordering cost functions $c_A(\sigma)$ and $c_B(\sigma)$ of the pertinent graph G_{μ_1} of μ_1 are not decreasing and convex; note that $c_A(\sigma)$ and $c_B(\sigma)$ coincide for $\sigma = 0$. By Property 2.2, μ_2 is either a Q^*-node or an S-node and its pertinent graph G_{μ_2} has an elbow-shaped cost-function by induction. Because of the elbow-shape of the cost function of G_{μ_2} and of the convexity of $c_A(\sigma)$ and $c_B(\sigma)$, an increase of one unit of spirality for the whole series causes an increase of at most one unit of the overall number of bends. These arguments imply that the two cost functions, (1) of the series of G_{μ_2} and G_{μ_1} with ordering A, and (2) of the series of G_{μ_2} and G_{μ_1} with ordering B, are elbow-shaped functions and have the same value for $\sigma = 0$. The cost function

of the whole series is the lower envelope of the two cost functions above; the lower envelope of two elbow-shaped functions that coincide for $\sigma = 0$ is an elbow-shaped function. □

6. Optimal orthogonal drawings of 3-planar graphs. The target of this section is to provide a polynomial time algorithm for computing the optimal orthogonal drawing of a 3-planar graph G. We first deal with the 2-connected case. The basic idea is to incrementally construct an optimal orthogonal drawing of G by composing orthogonal representations of its split components that are optimal within given values of spirality. To do that, we compute a decomposition tree T of G for each possible reference edge and equip the nodes of T with a data structure devised to describe optimal orthogonal representations of the split components of G. The optimal orthogonal drawing of G is selected among the set of optimal orthogonal drawings, each computed using one of the above decomposition trees.

In section 6.1 we present the data structure; in section 6.2 we show how to use T to compute optimal orthogonal drawings of 3-planar graphs. The management of several decomposition trees is shown in section 6.3. In section 6.4 we extend the algorithm to general 3-planar graphs.

6.1. Optimal sets. Let G be a 3-planar graph with n vertices, let T be a decomposition tree of G, and let μ be an internal node of T. Let G_μ be the pertinent graph of μ.

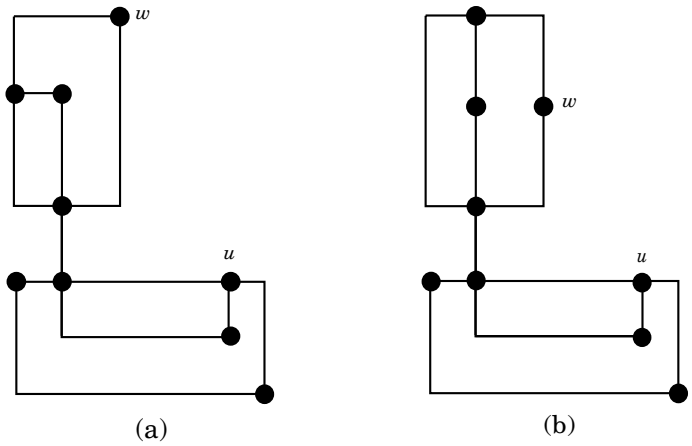


FIG. 21. *Feasible* (a) *and not feasible* (b) *orthogonal representation of the split component highlighted in Fig.* 6.

An orthogonal representation H_μ of G_μ is *feasible* if there exists an orthogonal representation of G having H_μ as a subgraph. As an example, in Fig. 21 two orthogonal representations of the split component in evidence in Fig. 6 are shown; the one in Fig. 21a is feasible while the one in Fig. 21b is not feasible. Notice that the feasibility of an orthogonal representation can be checked by looking at the labels in the adjacency lists of the poles and at their external degree.

Let H_μ be a feasible orthogonal representation of G_μ; let $\mathcal{H}$ be the infinite class of orthogonal representations of G such that (1) each element of $\mathcal{H}$ has H_μ as a subgraph; (2) the labels associated with the right (left) face of H_μ in the adjacency lists of its poles are the same for each element of $\mathcal{H}$. We say that $\mathcal{H}$ is a *fitting class* of H_μ.

Let u and w be the poles of H_μ. Let $\{\lambda_u, \rho_u, \lambda_w, \rho_w\}$ be a set of labels in $\{R, L\}$. The set $\{\lambda_u, \rho_u, \lambda_w, \rho_w\}$ is *sound* for H_μ if there exists a fitting class $\mathcal{H}$ of H_μ such that for each element of $\mathcal{H}$ the labels associated with the left (right) face of H_μ in the adjacency lists of u and w are λ_u (ρ_u) and λ_w (ρ_w).

Observe that a feasible orthogonal representation H_μ and a set of labels that are sound for H_μ identify a fitting class of H_μ. Also, by Property 4.2, H_μ and a set of labels that are sound for H_μ univocally determine the spirality of H_μ.

An *optimal 7-tuple* of μ is a 7-tuple $\langle H_\mu, \sigma_{H_\mu}, c_{H_\mu}, \lambda_u, \rho_u, \lambda_w, \rho_w \rangle$ such that

1. H_μ is a feasible orthogonal representation of G_μ, optimal within spirality σ_{H_μ};
2. c_{H_μ} is the cost of H_μ;
3. $\{\lambda_u, \rho_u, \lambda_w, \rho_w\}$ is a set of labels that is sound for H_μ.

We equip each node μ of T with an *optimal set*. An optimal set is a set of optimal 7-tuples of μ with the following properties.

1. For each value of spirality the set contains at most one optimal 7-tuple.
2. Let $\langle H_\mu, \sigma_{H_\mu}, c_{H_\mu}, \lambda_u, \rho_u, \lambda_w, \rho_w \rangle$ be an optimal 7-tuple of the optimal set. Then, for each node ν of the subtree of T rooted at μ (including μ), the orthogonal representation $H_\nu \subseteq H_\mu$ of the pertinent graph of ν has spirality in the range stated by Theorem 4.4.

Note that, by Corollary 4.1, the cardinality of an optimal set is at most $2(3n-2)+1$ if G_μ is even, and at most $2(3n-2)$ if G_μ is odd.

LEMMA 6.1. *There exists an optimal orthogonal representation H of G such that the split components of H belong to the optimal sets of the nodes of T.*

Proof. Let H' be any optimal orthogonal representation of G, and let $H'_\mu \subset H'$ be the orthogonal representation of the pertinent graph of an internal node μ of T. We show how to compute H starting from H'.

Let $\sigma_{H'_\mu}$ be the spirality of H'_μ. By Theorem 4.4, $|\sigma_{H'_\mu}| \leq 3n-2$; also, by definition, there is an orthogonal representation H''_μ that belongs to the optimal set of μ and is optimal within spirality $\sigma_{H'_\mu}$. Let H'' be an orthogonal representation in the fitting class of H''_μ such that H' and H'' are split-different for the split components H'_μ and H''_μ. We apply Theorem 4.1 to H' and H'', substituting H'_μ with H''_μ in H'. Since H''_μ is optimal within the spirality $\sigma_{H'_\mu}$, the orthogonal representation obtained with the substitution is still an optimal orthogonal representation of G. The optimal orthogonal representation H is obtained by iterating such a procedure for each split component of H'. □

6.2. Computing the optimal sets of 3-planar graphs. We consider separately the different types of internal nodes of a decomposition tree of a 3-planar graph G with n vertices and show how to compute the optimal sets of such nodes.

6.2.1. Computing the optimal set of a Q^*-node. Let μ be a Q^*-node of T, and let $G_\mu \subset G$ be the pertinent graph of μ with poles u and w. An optimal set of μ is computed by the following procedure. Intuitively, while increasing the spirality of a chain with $k+1$ vertices, we avoid bending the edges until spirality $k-1$; after that value we add one bend for each increasing unit of spirality.

PROCEDURE Q^*-OPTIMALSET(μ)
  for each integer σ between $-3n+2$ and $3n-2$, compute the 7-tuple $\langle H_\mu, \sigma_{H_\mu}, c_{H_\mu}, \lambda_u, \rho_u, \lambda_w, \rho_w \rangle$ as follows and put it in the optimal set of μ:
    Let k be the number of edges of G_μ.
    if $|\sigma| < k$ **then** H_μ has no bends and $c_{H_\mu} = 0$

else H_μ has $|\sigma| - k + 1$ bends and $c_{H_\mu} = |\sigma| - k + 1$.
$\sigma_{H_\mu} = \sigma$; labels λ_u, ρ_u, λ_w, ρ_w are undefined.
end procedure.

As an example, some 7-tuples of the optimal set of node ξ_1 of Fig. 3 computed by Procedure Q^*-OptimalSet are shown in Fig. 22a.

Since μ is a Q^*-node, an efficient way to encode H_μ is to store in each 7-tuple only the number of turns that appear in H_μ. By Corollary 4.1 and since each optimal 7 tuple is computed in $O(1)$ time, we have the following lemma.

LEMMA 6.2. *Procedure Q^*-OptimalSet computes an optimal set of a Q^*-node in $O(n)$ time.*

6.2.2. Computing the optimal set of an S-node. Let μ be an S-node of T, and let $G_\mu \subset G$ be the pertinent graph of μ with poles u and w. Let μ_1 and μ_2 be the two children of μ. Let G_{μ_1} be the pertinent graph of μ_1 with poles $u_1 \equiv u$ and w_1; let G_{μ_2} be the pertinent graph of μ_2 with poles $u_2 \equiv w_1$ and $w_2 \equiv w$. An optimal set of μ is computed by the following procedure. Intuitively, for each value of spirality stated by Corollary 4.1, we look into the optimal sets of μ_1 and μ_2 searching for the minimum cost 7-tuples that satisfy Lemma 4.2.

PROCEDURE S-OPTIMALSET(μ)
for each value of spirality stated by Corollary 4.1, compute the 7-tuple $\langle H_\mu, \sigma_{H_\mu}, c_{H_\mu}, \lambda_u, \rho_u, \lambda_w, \rho_w \rangle$ by performing the following steps and, if such a tuple exists, put it in the optimal set of μ:
Step 1: Find two 7-tuples t_1 and t_2 in the optimal sets of μ_1 and μ_2 such that:
(1) $t_1 = \langle H_{\mu_1}, \sigma_{H_{\mu_1}}, c_{H_{\mu_1}}, \lambda_{u_1}, \rho_{u_1}, \lambda_{w_1}, \rho_{w_1} \rangle$;
(2) $t_2 = \langle H_{\mu_2}, \sigma_{H_{\mu_2}}, c_{H_{\mu_2}}, \lambda_{u_2}, \rho_{u_2}, \lambda_{w_2}, \rho_{w_2} \rangle$;
(3) $\sigma_{H_{\mu_1}} + \sigma_{H_{\mu_2}} = \sigma$ (see Lemma 4.2); and
(4) $c_{H_{\mu_1}} + c_{H_{\mu_2}}$ is the lowest among all the possible choices of such 7-tuples.
if such 7-tuples do not exist **then** skip the next two steps.
Step 2: Define H_μ as follows.
• H_μ has the same vertices and edges of G_μ.
• Associate with each vertex of H_μ but $w_1 \equiv u_2$ the same adjacency list it has in H_{μ_1} or in H_{μ_2}.
• Compose the adjacency list L of $w_1 \equiv u_2$ as follows. Let L_1 (L_2) be the adjacency list of w_1 in H_{μ_1} (u_2 in H_{μ_2}). Cut L_1 and L_2 in correspondence with f_e, deleting f_e. Build the new list L as the concatenation of L_1, f_e, L_2, and f_e. Associate with the first (second) occurrence of f_e the label λ_{w_1} (ρ_{w_1}) of t_1 if u_2 is a bridge pole in G_{μ_2}, substitute the label λ_{u_2} (ρ_{u_2}) of t_2 otherwise. Make L circular by linking its first element to its last element.
Step 3: $\lambda_u = \lambda_{u_1}$, $\rho_u = \rho_{u_1}$, $\lambda_w = \lambda_{w_2}$, $\rho_w = \rho_{w_2}$. $c_{H_\mu} = c_{H_{\mu_1}} + c_{H_{\mu_2}}$.
end procedure.

As an example, some 7-tuples of the optimal set of node ξ_2 of Fig. 3 computed by Procedure S-OptimalSet are shown in Fig. 22b.

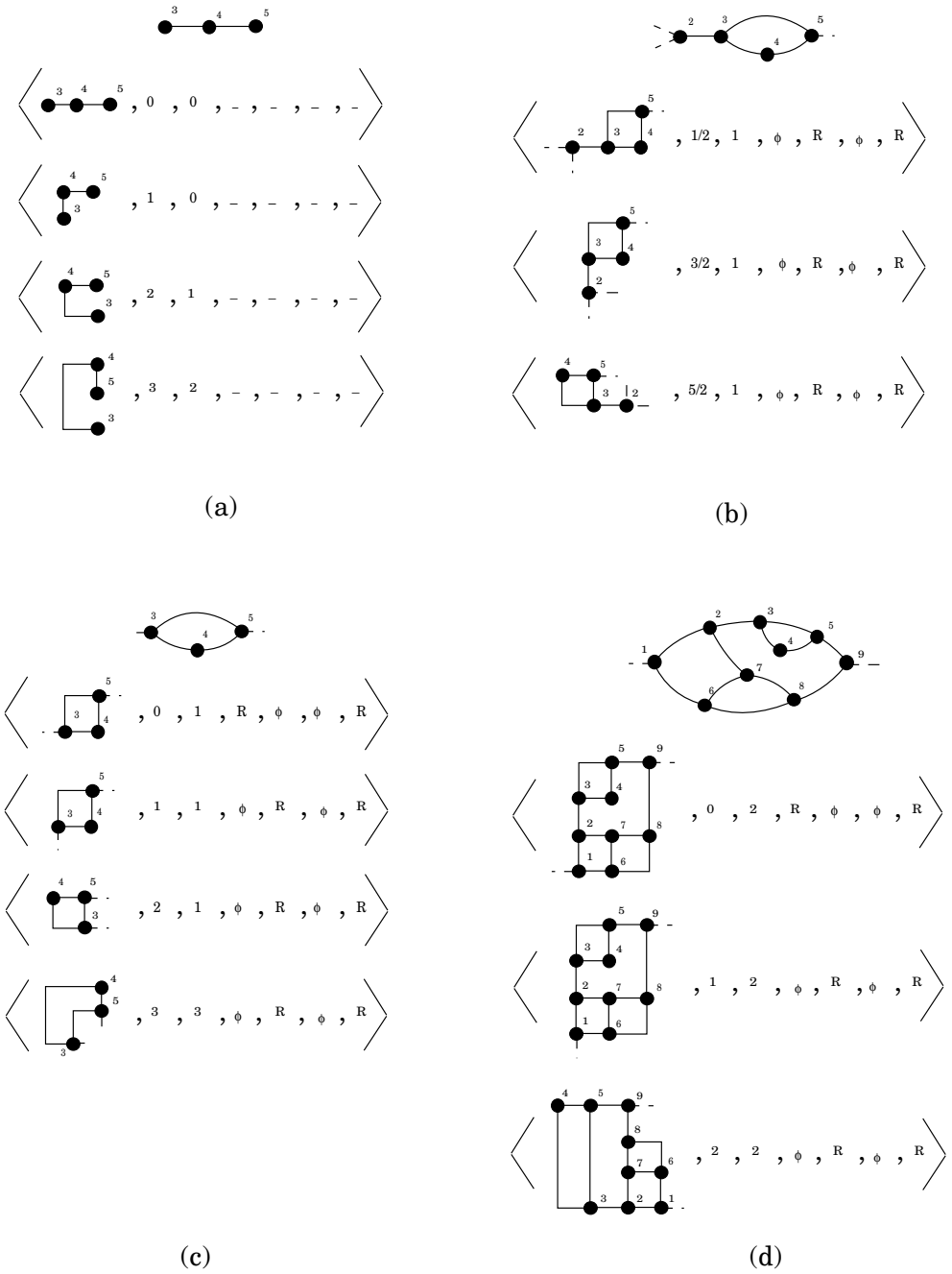


FIG. 22. *Some optimal* 7*-tuples of the nodes:* (a) ξ_1, (b) ξ_2, (c) ξ_3, *and* (d) ξ_4 *of the decomposition tree of Fig.* 3.

LEMMA 6.3. *Procedure S-OptimalSet computes an optimal set of an internal S-node μ in $O(n^2)$ time.*

Proof. We concentrate on each computed 7-tuple.

We first prove that H_μ is a feasible orthogonal representation of G_μ. We have that H_μ is an orthogonal representation by construction. Also, since poles u and w coincide with poles u_1 and w_2 and both H_{μ_1} and H_{μ_2} are feasible, H_μ is feasible.

Second, suppose H_μ is not optimal within spirality σ_{H_μ}; then there exists a feasible orthogonal representation H'_μ of G_μ with fewer bends than H_μ and spirality σ_{H_μ}, and such that, by Lemma 6.1, all the split components of H'_μ are in the 7-tuples of the optimal sets of the children of node μ. Since for each value of spirality Procedure S-OptimalSet considers all the possible pairs of 7-tuples in the optimal sets of the children of μ, we have a contradiction.

Third, we observe that since the labels of t_1 and t_2 are sound, labels λ_u, λ_w, ρ_u, and ρ_w are sound for H_μ.

To complete the proof of correctness, it remains to prove that the set defined by Procedure S-OptimalSet is an optimal set of μ. This easily follows from the construction strategy and from the fact that the procedure considers optimal sets of μ_1 and μ_2.

Concerning the complexity, since μ is an S-node, an efficient way to encode H_μ in the 7-tuple is to store two pointers to t_1 and t_2 plus the labels of the faces around the poles of H_μ. Thus, the computation of an optimal 7-tuple can be performed in linear time by using sorted structures for the optimal sets of the children of μ. So, we have that Steps 1–3 compute an optimal 7-tuple of an S-node μ for a given value σ of spirality in $O(n)$ time. The overall time complexity follows from Corollary 4.1. □

6.2.3. Computing the optimal set of a P-node. If μ is a P-node, from Property 2.2 it has two children. We give a procedure to compute an optimal set of μ. The procedure exploits the result of Lemma 4.4. Namely, we consider all possible values of parameters α_v^i $(v = u, w)$ in any orthogonal representation of G containing H_μ; α_v^1 and α_v^2 can be either 0 or 1, but they cannot both be 0.

PROCEDURE P-OPTIMALSET(μ)

for each value of spirality stated by Corollary 4.1, compute the 7-tuple $\langle H_\mu, \sigma_{H_\mu}, c_{H_\mu}, \lambda_u, \rho_u, \lambda_w, \rho_w \rangle$ by performing the following steps and, if such a tuple exists, put it in the optimal set of μ:

Step 1: Find a 7-tuple in the optimal set of μ_1, and a 7-tuple in the optimal set of μ_2 such that:

(1) the spiralities in each one of the two selected 7-tuples satisfy formula of Lemma 4.4 with an admissible value for α_v^i;

(2) the sum of the costs is the lowest among all the possible choices of the two 7-tuples.

if such two 7-tuples do not exist **then** skip the next two steps.

Step 2: Let $t_1 = \langle H_1, \sigma_{H_1}, c_{H_1}, \lambda_{u_1}, \rho_{u_1}, \lambda_{w_1}, \rho_{w_1} \rangle$ and $t_2 = \langle H_2, \sigma_{H_2}, c_{H_2}, \lambda_{u_2}, \rho_{u_2}, \lambda_{w_2}, \rho_{w_2} \rangle$ be the 7-tuples chosen in Step 1. Let $\sigma_{H_\mu} = \sigma_{H_1} + \alpha_u^1 + \alpha_w^1$ and $\sigma_{H_\mu} = \sigma_{H_2} + \alpha_u^2 + \alpha_w^2$, (observe that we have removed parameters k_u and k_w in the formula of Lemma 4.4, since the degree of poles u and w is 3).

Define H_μ as follows (see Fig. 23).

- H_μ has the same vertices and edges of G_μ.

• Associate to each vertex of H_μ but u and w the same adjacency list it has in either H_1 or H_2.

• Let e and e' be the edges of H_1 and of H_2 that are incident on pole u, respectively. Define the adjacency list of u as edge e and edge e'. Similarly, define the adjacency list of w. Let f be the face of H_μ composed by vertices and edges of H_1 and H_2. Assign an R-label to f in the adjacency list of v if $\alpha_v^1 + \alpha_v^2 = 1$ $(v = u, w)$; assign no label to f otherwise. Assign an L-label to the external face f_e in the adjacency list of v if $\alpha_v^1 + \alpha_v^2 = 1$; assign no label to f_e otherwise.

Step 3: **if** $\alpha_v^1 = 1$, **then** assign an R-symbol to λ_v $(v = u, w)$, **else** assign no symbol to λ_v. **if** $\alpha_v^2 = 1$, **then** assign an R-symbol to ρ_v, **else** assign no symbol to ρ_v. $c_{H_\mu} = c_{H_1} + c_{H_2}$.

end procedure.

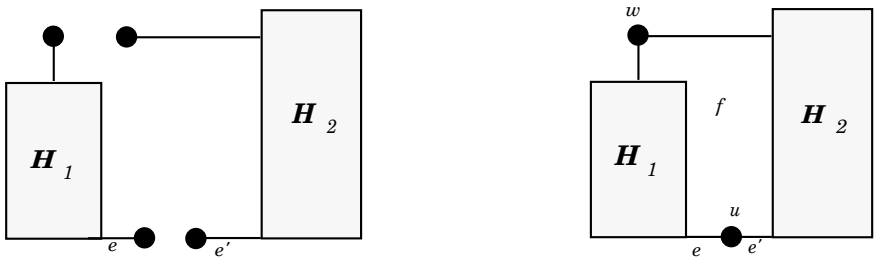


FIG. 23. *Illustration for Procedure P-OptimalSet.*

As an example, some 7-tuples of the optimal set of node ξ_3 of Fig. 3 computed by Procedure P-OptimalSet are shown in Fig. 22c.

LEMMA 6.4. *Procedure P-OptimalSet computes an optimal set of an internal P-node μ with two children in $O(n)$ time.*

Proof. We concentrate on each computed 7-tuple.

We first prove that H_μ is a feasible orthogonal representation of G_μ. We need to prove that the conditions of Property 2.3 hold. Condition 1 is trivially true by construction. Condition 2 is trivially true for all faces but f and the external face. Consider f. Its boundary is composed by a spine S_1 of H_1, and a spine S_2 of H_2. Also, by construction, an R-label is assigned to f in the adjacency list of v (w) if $\alpha_v^1 + \alpha_v^2 = 1$ $(\alpha_w^1 + \alpha_w^2 = 1)$.

Thus, writing the number of R-labels on both the poles as $2-\alpha_u^1-\alpha_u^2+2-\alpha_w^1-\alpha_w^2$ and using Lemma 4.4, the number of right turns minus the number of left turns encountered when going around f in the positive direction is $n(S_2)-n(S_1)+2-\alpha_u^1-\alpha_u^2+2-\alpha_w^1-\alpha_w^2 = \alpha_u^1+\alpha_u^2+\alpha_w^1+\alpha_w^2+2-\alpha_u^1-\alpha_u^2+2-\alpha_w^1-\alpha_w^2 = 4$. Analogous reasoning for the external face.

Since the external degree of both the poles is 1 and the external face has no R-labels in the adjacency lists of the poles, H_μ is feasible. λ_v and ρ_v $(v = u, w)$ are sound for H_μ since by construction they cannot both correspond to no label.

H_μ is optimal within spirality σ_{H_μ}: the proof is analogous to that of Lemma 6.3.

With analogous reasoning as in Lemma 6.3, it can be proved that the set defined by Procedure P-OptimalSet is an optimal set of μ.

Concerning the complexity, since μ is a P-node, an efficient way to encode H_μ is to store only two pointers to the 7-tuples that have been chosen as well as the labels

of the faces around the poles of H_μ. Thus, the computation of an optimal 7-tuple can be performed in constant time by using sorted structures for the optimal sets of the children of μ. So, we have that Steps 1–3 compute an optimal 7-tuple of a P-node μ with two children for a given value σ of spirality in $O(1)$ time. The overall time complexity follows from Corollary 4.1. □

6.2.4. Computing the optimal set of an R-node. Let μ be an R-node of T and let $G_\mu \subset G$ be the pertinent graph of μ with poles u and w. An optimal set of μ is computed by the following procedure. Intuitively, for each value of spirality, (1) we compute the two flow networks associated with ordering A and ordering B, (2) we solve the two corresponding min-cost flow problems, (3) we choose the minimum cost solution between them, and (4) we construct the 7-tuple corresponding to the selected solution.

PROCEDURE R-OPTIMALSET (μ)

Define ordering A and ordering B of G_μ.

for each value of spirality stated by Corollary 4.1, compute the 7-tuple $\langle H_\mu, \sigma_{H_\mu}, c_{H_\mu}, \lambda_u, \rho_u, \lambda_w, \rho_w \rangle$ by performing the following steps and, if such a tuple exists, put it in the optimal set of μ:

Step 1:

- Construct $N_\mu(\sigma_{H_\mu}, A)$ and $N_\mu(\sigma_{H_\mu}, B)$.
- Solve the min-cost flow problem on $N_\mu(\sigma_{H_\mu}, A)$ and on $N_\mu(\sigma_{H_\mu}, B)$. Choose the network whose solution has the minimum cost.
- Let ν be a child of μ in T, whose pertinent graph is G_ν. Check if it is possible to construct an orthogonal representation H_μ of G_μ by applying the technique of Lemma 5.8 and by using for each component G_ν only the orthogonal representations in the optimal set of ν.

if the construction fails **then** skip the next step.

Step 2:

Let H'_μ be the orthogonal representation of *skeleton*(μ) constructed by the network chosen in Step 1. (See Lemma 5.8.)

Let f_{uw} be the face of H'_μ sharing (u, w) with the external face f_e.

- Assign to λ_u and ρ_u the labels of f_e and f_{uw}, respectively, in the adjacency list of u in H'_μ.
- Assign to λ_w and ρ_w the labels of f_e and f_{uw}, respectively, in the adjacency list of w in H'_μ.
- Assign to c_{H_μ} the cost of the flow.

end procedure.

As an example, some 7-tuples of the optimal set of node ξ_4 of Fig. 3 computed by Procedure R-OptimalSet are shown in Fig. 22d.

LEMMA 6.5. *Procedure R-OptimalSet computes an optimal set of an R-node in $O(n^3 \log n)$ time.*

Proof. We concentrate on each computed 7-tuple.

The proof that H_μ is a feasible orthogonal representation of G_μ, optimal within spirality σ_{H_μ}, directly descends from Lemma 5.8.

Labels λ_u, λ_w, ρ_u, and ρ_w are sound for H_μ by construction.

To complete the proof of correctness, it remains to prove that the set defined by Procedure S-OptimalSet is an optimal set of μ. This easily follows from the construction strategy and from the fact that the procedure considers only optimal sets of the children of μ.

Concerning the complexity, since the cost functions of the arcs of the network are elbow-shaped functions (see Theorem 5.1), it is possible to solve the min-cost flow problem by means of the Out-of-Kilter algorithm [15] whose time complexity is $O(n^2 \log n)$. The construction of H_μ from the flow network can be easily performed in linear time. The overall time complexity follows from Corollary 4.1. □

6.3. The drawing algorithm. In this section we show that it is possible to compute in polynomial time the optimal sets of the internal nodes of a decomposition tree of a 3-planar graph. Then, we give an algorithm that computes an optimal orthogonal drawing of a 3-planar graph by considering all its possible embeddings.

Let G be a 3-planar graph with n vertices, and let T be a decomposition tree of G. The following procedure equips each internal node of T with an optimal set, by means of a bottom-up visit of T.

PROCEDURE EQUIP-$T(\mu)$
Consider with a bottom-up visit each nonroot node μ of T
case (μ) **of**
Q^*-node: apply Procedure Q^*-OptimalSet(μ),
P-node: apply Procedure P-OptimalSet(μ),
S-node: apply Procedure S-OptimalSet(μ),
R-node: apply Procedure R-OptimalSet(μ),
end procedure.

LEMMA 6.6. *Procedure Equip-T equips the internal nodes of T with optimal sets in $O(|P|n + |Q|n + |S|n^2 + |R|n^3 \log n)$ time, where $|P|$, $|Q|$, $|S|$, and $|R|$ denote the number of internal P-nodes, Q^*-nodes, S-nodes, and R-nodes of T, respectively.*

Proof. The proof directly descends from Lemmas 6.2, 6.3, 6.4, and 6.5. □

We are now at a mature stage for computing an optimal orthogonal representation of G. The next procedure exploits the result of Lemma 4.5. Namely, we consider all possible values of the parameter α_v $(v = s, t)$ in an orthogonal representation of G. Such possible values depend on the degree of v: if $\deg(v) = 3$ the value of α_v can be either 0 or 1; otherwise, if $\deg(v) = 2$, the value of α_v can be -1, 0, or 1.

PROCEDURE H-from-T
Let μ be the root of T, let μ_1 and μ_2 be the children of μ; the pertinent graph of μ_2 is the reference edge (s, t).
Step 1: Apply Procedure Equip-$T(\mu)$.
Step 2: Find two 7-tuples t_1 and t_2 in the optimal sets of μ_1 and μ_2 such that:
(1) $t_1 = \langle H_{\mu_1}, \sigma_{H_{\mu_1}}, c_{H_{\mu_1}}, \lambda_{u_1}, \rho_{u_1}, \lambda_{w_1}, \rho_{w_1} \rangle$;
(2) $t_2 = \langle H_{\mu_2}, \sigma_{H_{\mu_2}}, c_{H_{\mu_2}}, \lambda_{u_2}, \rho_{u_2}, \lambda_{w_2}, \rho_{w_2} \rangle$;
(3) $\sigma_{H_{\mu_1}}$ and $\sigma_{H_{\mu_2}}$ satisfy Lemma 4.5; and
(4) $c_{H_{\mu_1}} + c_{H_{\mu_2}}$ is the lowest among all the possible choices of such 7-tuples.
Step 3: Define H as follows.
• Associate with each vertex of H but s and t the same adjacency list it has in H_{μ_1}.

• Associate with edge (s,t) the same sequence of L- and R-symbols it has in H_{μ_2}. Let e_1 and e_2 be the (possibly coincident) edges of H_{μ_1} incident on s. Associate with s in H the same adjacency list it has in H_{μ_1}; insert edge (s,t) in the adjacency list of s between e_1 and e_2. Do analogously for vertex t. Let f_e be the external face of H and let f be the internal face of H that contains (s,t). Associate labels to f_e and to f in the adjacency list of pole v ($v = s,t$) with the following rule.

if $\deg(v) = 2$, **then**

if $\alpha_v = -1$ **then** assign an R-label to f_e and an L-label to f

if $\alpha_v = 0$ **then** assign no label to either f_e or f

if $\alpha_v = 1$ **then** assign an L-label to f_e and an R-label to f

else assign λ_{v_1} to f_e and ρ_{v_1} to f.

end procedure.

LEMMA 6.7. *Procedure H-from-T computes an optimal orthogonal representation of G in $O(|P|n+|Q|n+|S|n^2+|R|n^3\log n)$ time, where $|P|$, $|Q|$, $|S|$, and $|R|$ denote the number of P-nodes, Q^*-nodes, S-nodes, and R-nodes of T, respectively.*

Proof. The proof that H is an orthogonal representation of G can be done with reasoning analogous to that in the proof of Lemma 7.1.

Also, H_μ is optimal since Procedure H-from-T considers all the possible pairs of 7-tuples in the optimal set of μ_1 and μ_2 (see also Lemma 6.1).

Concerning the complexity, note that (1) Step 1 requires $O(|P|n+|Q|n+|S|n^2+|R|n^3\log n)$ (see Lemma 6.6), (2) Step 2 requires $O(n)$, and (3) Step 3 requires $O(n)$ to construct H; namely, since the orthogonal representations are stored in the optimal sets by means of pointers, the procedure has to decode such information in order to construct H. This task can be performed in linear time. The bound on the time complexity follows. □

We now give the algorithm for optimal orthogonal drawing 2-connected 3-planar graphs; G denotes the input graph.

ALGORITHM OPTIMAL ORTHOGONAL DRAWING

Step 1: for each edge (u,w) of G:

• Compute a decomposition tree T of G with reference edge (u,w).

• Compute an orthogonal representation applying Procedure H-from-T.

Step 2: Choose, among the ones computed in the first step, the orthogonal representation H with minimum cost, and construct from H an optimal drawing of G.

end Algorithm.

THEOREM 6.1. *Algorithm Optimal Orthogonal Drawing computes an optimal orthogonal drawing of a 2-connected 3-planar graph with n vertices in $O(n^5\log n)$ time.*

Proof. For each possible choice of the reference edge (s,t), the algorithm computes an optimal orthogonal representation by implicitly considering all possible embeddings with (s,t) on the external face (Lemma 6.7). This makes the algorithm consider all the embeddings of G.

Concerning the time complexity of Step 1 of the algorithm observe that (1) the computation of a decomposition tree requires $O(n)$ time [5] and (2) Procedure H-from-T requires $O(n^4\log n)$ time (Lemma 6.7). Finally, Step 2 can be performed in $O(n)$ time by using the compaction algorithm presented in [21, 25]. The time complexity bound follows. □

6.4. Extension to general 3-planar graphs. It is easy to extend to general 3-planar graphs the result of Theorem 6.1. Namely, we can give the following theorem.

THEOREM 6.2. *There exists an algorithm that computes an optimal orthogonal drawing of a 3-planar graph with n vertices in $O(n^5 \log n)$ time.*

Proof. We exploit the technique illustrated in [14] (section 11.2.3) that is based on (1) computing a block tree T_{BC} that describes a decomposition of a 1-connected graph into its *blocks* (2-connected components), and (2) assembling the orthogonal representations of such blocks by bottom-up following the structure of T_{BC}. The theorem follows from the properties of the block trees and from the following fact that can be used for each block.

Let v be a degree-2 vertex of the external face f of an optimal orthogonal drawing of a 3-planar 2-connected graph and let (u, v) and (v, w) be the two edges incident on v. The label associated with v in f cannot be an R. In other words, the angle between (u, v) and (v, w) on f is not convex.

This fact is easily proved by using Lemma 5.2 on the split components identified by the separation pair u, w. □

7. Extension to optimal orthogonal drawings of series-parallel graphs. In this section we extend the above results on 3-planar graphs to (4-planar) series-parallel graphs. Let G be a series-parallel graph and let T be the decomposition tree of G. Clearly, the computation of the optimal sets of the Q^*-nodes and of the S-nodes of T can be performed with the procedures of the previous section. Concerning P-nodes, if they have two children and both the poles have degree 3, then we can apply Procedure P-OptimalSet; otherwise we have to use a slightly different technique. Of course, because of [11], it is unlikely that the computation of the optimal sets of R-nodes of 4-planar graphs can be performed in polynomial time.

Let μ be an internal P-node of T and let $G_\mu \subset G$ be the pertinent graph of μ with poles u and w. Let μ_i $(i = 1 \dots k)$ be the children of μ. An optimal set of μ can be computed extending Procedure P-OptimalSet by exploiting both Lemmas 4.3 and 4.4. The extensions are the following.

- If $k = 2$ and u (or w) has degree 4, then when constructing H_μ (see Step 2 of P-OptimalSet) the adjacency list of u is defined as follows. Let e_1 and e_2 be the edges (possibly coincident) of H_1 incident on pole u such that, when going around the external face of H_1 in the positive direction, e_1, u, and e_2 appear consecutively. Let e'_1 and e'_2 be the edges (possibly coincident) of H_2 incident on pole u such that, when going around the external face of H_2 in the positive direction, e'_1, u, and e'_2 appear consecutively. Edges e_1, e_2, e'_1, and e'_2 are assigned with this clockwise order around u in H_μ. Let f_a be the face of H_μ composed by vertices and edges of H_1 and H_2. Assign an R-label to f_a in the adjacency list of v, if $\deg(v) = 4$, or $\deg(v) = 3$ and $\alpha_v^1 + \alpha_v^2 = 1$ $(v = u, w)$; assign no label to f_a otherwise. Assign an L-label to the external face f_e in the adjacency list of v if $\alpha_v^1 + \alpha_v^2 = 1$ or if the external degree of v in G_μ is 2; assign no label to f_e otherwise.
- If $k = 3$, then we use Lemma 4.3 for selecting the optimal 7-tuples (see Step 1 of P-OptimalSet). The construction of H_μ is similar to the one above.

The proof of the following lemma is a variation of the one of Lemma 6.4.

LEMMA 7.1. *There exists a procedure that computes an optimal set of an internal P-node μ in $O(n)$ time.*

By using the drawing algorithm of the previous section and the above lemma, we have the following theorem. The time bound can be proved with a technique similar to that of Theorem 6.1.

THEOREM 7.1. *There exists an algorithm that computes an optimal orthogonal drawing of a 4-planar series-parallel graph with n vertices in $O(n^4)$ time.*

It is worth noting that the bound of Theorem 7.1 can be improved when the input graph is a 3-planar series-parallel graph. The proof of the following theorem can be found in [4].

THEOREM 7.2. *There exists an algorithm for computing an optimal orthogonal drawing of a 3-planar series-parallel graph with n vertices in $O(n^3)$ time.*

8. Conclusions and open problems. In this paper we have given a general theory of the interplay between spirality and orthogonal drawings and we have shown that the problem of finding the embedding of a graph that underlies its optimal orthogonal drawing can be solved in polynomial time for series-parallel graphs and 3-planar graphs.

Although the problem for general graphs is NP-complete [11], several questions are open.

- Devise an algorithm for optimal orthogonal drawings of 4-planar graphs whose time complexity is exponential in the number of vertices of degree 4.
- The emphasis of this paper is on the existence of polynomial time algorithms. However, the time complexity of the algorithms presented in the paper could perhaps be improved. For example, one can think of applying techniques similar to those of [20].
- Several of the algorithms presented in the paper can be easily parallelized by using standard techniques. However, the bound that can be achieved with such techniques is not optimal. The problem of finding efficient parallel algorithms still exists.

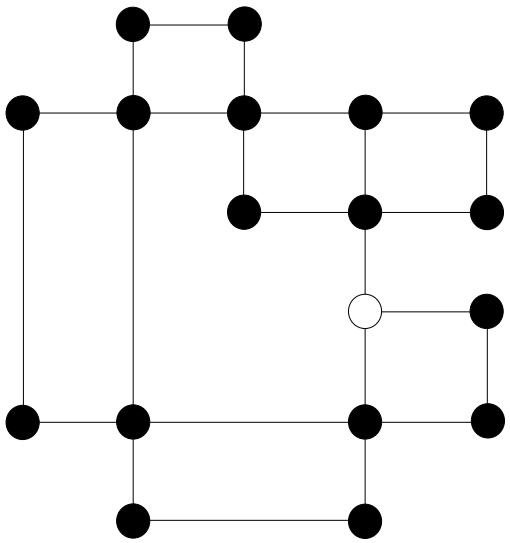


FIG. 24. *A problem in extending Theorem* 6.2 *to series-parallel graphs.*

- Extend our results on series-parallel graphs to the non-2-connected case. Exploiting the technique of [14] seems to be harder in this case than in Theorem 6.2. Figure 24 shows an optimal orthogonal drawing of a 2-connected

series-parallel graph that contains a vertex (the white one) of degree 3 whose angle on the external face is convex. If that vertex is a cutvertex of a non-2-connected series-parallel graph, then the technique of [14] cannot be directly applied.

Appendix. List of defined terms.

We list, in order of appearance, the main terms not defined in the preliminaries. Beside each term we give the section where it is defined.

Term	Section
bridge pole	4.1
nonbridge pole	4.1
alias vertex	4.1
spine	4.1
spirality	4.2
split-different	4.5
substitution	4.5
optimality within a given spirality	4.6
odd split component	4.6
even split component	4.6
cost function of a split component	4.7
fixed-embedding cost function of a split component	4.7
access edge	5.1
starting face	5.1
target face	5.1
access vertex	5.1
access	5.1
equivalent curves	5.1
ring	5.1
nonincreasing ring	5.1
elbow-shaped function	5.2
fixed-ordering cost function	5.2
ordering A	5.2
ordering B	5.2
feasibility of an orthogonal representation	6.1
fitting class	6.1
optimal 7-tuple	6.1
optimal set	6.1

Acknowledgments. We thank Roberto Tamassia for his helpful comments and suggestions. We thank an anonymous referee and Goos Kant for suggesting to us the extension to 1-connected 3-planar graphs. We also thank the anonymous referee for all the suggestions given to improve the readability of the paper.

REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows Theory, Algorithms, and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[2] P. Bertolazzi, R. F. Cohen, G. Di Battista, R. Tamassia, and I. G. Tollis, *How to draw a series-parallel digraph*, in Proc. 3rd Scandinavian Workshop on Algorithm Theory, 1992, pp. 272–283; Internat. J. Comput. Geom. Appl., 4 (1994), pp. 385–402.

[3] T. Biedl and G. Kant, *A better heuristic for orthogonal graph drawings*, in Algorithms - ESA'94, Second Annual European Symposium, Utrecht, 1994, J. van Leeuwen, ed., Lecture Notes in Comput. Sci. 855, Springer-Verlag, New York, 1994, pp. 24–35.

[4] G. Di Battista, G. Liotta, and F. Vargiu, *Spirality and Optimal Orthogonal Drawings*, Tech. Report 07.94, Dipartimento di Informatica e Sistemistica, Universita' di Roma "La Sapienza," 1994.

[5] G. Di Battista and R. Tamassia, *Incremental planarity testing*, in Proc. 30th IEEE Symp. on Foundations of Computer Science, IEEE, Piscataway, NJ, 1989, pp. 436–441.

[6] G. Di Battista and R. Tamassia, *On Line Planarity Testing*, Tech. Report CS-89-31, Dept. of Computer Science, Brown Univ., Providence, RI, 1989.

[7] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Algorithms for automatic graph Drawing: An annotated bibliography*, Comput. Geom. Theory Appl., 4 (1994), pp. 235–282.

[8] S. Even, *Graph Algorithms*, Computer Science Press, Potomac, MD, 1979.

[9] H. Everett and D. G. Corneil, *Recognizing visibility graphs of spiral polygons*, J. Algorithms, 11 (1990), pp. 1–26.

[10] D. R. Fulkerson, *An out-of-kilter method for minimal cost flow problems*, SIAM J. Appl. Math., 9 (1961), pp. 18–27.

[11] A. Garg and R. Tamassia, *On the computational complexity of upward and rectilinear planarity testing*, in Graph Drawing, Dimacs Internat. Workshop, GD'94, Princeton, NJ, 1994, R. Tamassia and I. G. Tollis, eds., Lecture Notes in Comput. Sci. 894, Springer-Verlag, New York, 1994, pp. 286–297.

[12] J. Hopcroft and R. E. Tarjan, *Efficient planarity testing*, J. Assoc. Comput. Mach., 21 (1974), pp. 549–568.

[13] G. Kant, *Drawing planar graphs using the canonical-ordering*, in Proc. IEEE Symp. on Foundations of Computer Science, Pittsburgh, IEEE, Piscataway, NJ, 1992, pp. 101–110.

[14] G. Kant, *Algorithms for Drawing Planar Graphs*, Ph.D. Thesis, Dept. Computer Science, Univ. of Utrecht, the Netherlands, 1993.

[15] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[16] Y. Liu, A. Morgana, and B. Simeone, *A Linear Algorithm for* 3*-Bend Embeddings of Planar Graphs in the Grid*, manuscript, 1993.

[17] T. Nishizeki and N. Chiba, *Planar Graphs: Theory and Algorithms*, Ann. Discrete Math. 32, North-Holland, Amsterdam, 1988.

[18] A. Papakostas and I. G. Tollis, *Improved algorithms and bounds for orthogonal drawings*, in Graph Drawing, Dimacs Internat. Workshop, GD'94, Princeton, NJ, 1994, R. Tamassia and I. G. Tollis, eds., Lecture Notes in Comput. Sci. 894, Springer-Verlag, New York, 1994, pp. 40–51.

[19] J. A. Storer, *On minimal node-cost planar embeddings*, Networks, 14 (1984), pp. 181–212.

[20] K. Takamizawa, T. Nishizeki, and N. Saito, *Linear-time computability of combinatorial problems on series-parallel graphs*, J. Assoc. Comput. Mach., 29 (1982), pp. 623–641.

[21] R. Tamassia, *On embedding a graph in the grid with the minimum number of bends*, SIAM J. Comput., 16 (1987), pp. 421–444.

[22] R. Tamassia, *Planar orthogonal drawings of graphs*, in Proc. IEEE Internat. Symp. on Circuits and Systems, IEEE, Piscataway, NJ, 1990, pp. 319–322.

[23] R. Tamassia, G. Di Battista, and C. Batini, *Automatic graph drawing and readability of diagrams*, IEEE Trans. Systems, Man Cybernet., SMC-18 (1988), pp. 61–79.

[24] R. Tamassia and I. G. Tollis, *Efficient embedding of planar graphs in linear time*, in Proc. IEEE Internat. Symp. on Circuits and Systems, Philadelphia, IEEE, Piscataway, NJ, 1987, pp. 495–498.

[25] R. Tamassia and I. G. Tollis, *Planar grid embedding in linear time*, IEEE Trans. Circuits Systems, CAS-36 (1989), pp. 1230–1234.

[26] R. Tamassia, I. G. Tollis, and J. S. Vitter, *Lower bounds and parallel algorithms for planar orthogonal grid drawings*, in Proc. IEEE Symp. on Parallel and Distributed Processing, IEEE, Piscataway, NJ, 1991, pp. 386-393.

[27] R. Tamassia, I. G. Tollis, and J. S. Vitter, *Lower bounds for planar orthogonal drawings of graphs*, Inform. Process. Lett., 39 (1991), pp. 35–40.

[28] J. Valdes, R. E. Tarjan, and E. L. Lawler, *The recognition of series parallel digraphs*, SIAM J. Comput., 11 (1982), pp. 298–313.

[29] L. Valiant, *Universality considerations in VLSI circuits*, IEEE Trans. Computers, C-30 (1981), pp. 135–140.

[30] G. Vijayan and A. Wigderson, *Rectilinear graphs and their embeddings*, SIAM J. Comput., 14 (1985), pp. 355–372.